Rúben Filipe Pereira de Sousa

Avaliador de Exercícios Baseados em Diagramas

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Junho de 2015

Rúben Filipe Pereira de Sousa

Avaliador de Exercícios Baseados em Diagramas

Dissertação submetida à Faculdade de Ciências da Universidade do Porto como parte dos requisitos para a obtenção do grau de Mestre em Engenharia de Redes em Sistemas Informáticos

Orientador: Prof. Dr. José Paulo Leal

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Junho de 2015

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, o Prof. Dr. José Paulo Leal, pela ajuda que me deu ao longo deste projeto. O seu contributo foi, sem dúvida, essencial para a evolução desta dissertação. Agradeço imenso a disponibilidade que teve para me receber e por ter partilhado comigo os seus conhecimentos.

Gostaria também de agradecer à minha família, à minha namorada e aos meus amigos pelo apoio e pela força que me deram e continuam a dar, especialmente nas fases mais complicadas.

A todos eles, a minha infinita gratidão.

Resumo

Este projeto teve por objetivo principal a criação de um avaliador para comparar dois diagramas (solução e tentativa), de forma a identificar as suas diferenças e calcular uma classificação. Com essa nota e com esse conjunto de diferenças pretende-se ajudar alunos com respostas erradas.

Os diagramas são representações esquemáticas de informação que, ignorando o posicionamento dos seus elementos, podem ser abstraídas em grafos. É com base nesta noção que funciona o algoritmo, visto que está preparado para comparar dois grafos.

O algoritmo tem como principal função encontrar o mapeamento entre nós solução e tentativa que minimiza o conjunto de diferenças entre eles. Ao mesmo tempo pretende-se evitar o teste de todas as permutações possíveis, de forma a tornar o avaliador eficiente. Por isso, o algoritmo de comparação de grafos foi implementado de forma a testar em primeiro lugar os mapeamentos que possuem maior probabilidade de serem os corretos, permitindo, a certa altura, a aplicação de um critério de corte.

Para certificar o correto funcionamento do algoritmo, foi criado um gerador de grafos conexos com parâmetros configuráveis. Com esse gerador foi possível criar pares de grafos próximos com um conjunto de diferenças bem determinado. Assim, foi possível executar milhares de testes, que validaram o funcionamento do algoritmo, isto é, que permitiram verificar que o conjunto de diferenças e respetiva nota encontrada pelo avaliador são exatamente iguais ao esperado.

Para além disso foi criado um *parser* de forma a converter ficheiros com a representação dos diagramas em grafos estendidos. Este *parser* foi implementado para lidar com três tipos de diagramas: UML (diagramas de classes e de casos de uso) e EER. Este mecanismo usado na parte da criação de um experiência para ser utilizada com alunos reais porque permitiu, através de um editor, criar os seus próprio diagramas no computador. Posteriormente, esses diagramas puderam ser submetidos e analisados pelo avaliador, que foi integrado na plataforma *Mooshak*.

Abstract

This project has as main goal the creation of an assessor to compare two diagrams (solution and attempt)., in order to indentify their differences and compute a rating. With that grade and set of differences is intended to help students with wrong answers.

The diagrams are schematic representations of informations that, ignoring the layout, can be asbtract as graphs. It is based on this that tha algorithm is works, as it is prepared to compare two graphs.

The algorithm's main role is to find the map between solution and attempt nodes that minimizes the set of differences between them. At the same time, it is intended to avoid testing all possible permutations in order to make the assessor more efficient. Therefore, the graph comparison algorithm was implemented to check, in first place, the maps that have more probability of being the right ones, allowing, at one point, to prune.

To ensure the algorithm's correct behaviour, it was created a connected graphs generator with configurable parameters. Based on that generator, it was possible to generate pairs of close graphs with a well determined set of differences. Thus, it was possible to run thousands of tests, validating the algorithm's behaviour, i.e., which allowed to verify that the set of differences and grade found by the assessor are exactly the same as the expected.

In addition it was created a parser to convert the files with diagrams representations into extended graphs. This was implemented to deal with three kinds of diagrams: UML (class and use case) and EER. This was used to run an experiment with real students who, throuhg an editor, could create their own diagramas on computer. Then, the diagrams were submited and evaluated by the assessor, that was built in *Mooshak*.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Hoje em dia o ensino universitário é caracterizado pela existência de turmas com elevado número de alunos. Para além das turmas serem grandes, é frequente as unidades curriculares apresentarem um programa disciplinar extenso. Estes dois fatores fazem com que a carga laboral dos professores seja pesada, no que diz respeito à criação, entrega e a avaliação de exercícios [DLO05, dQ12]. Deste modo, nem sempre os professores têm disponibilidade para receberem e esclarecerem as dúvidas dos seus alunos.

Durante as aulas são expostos os conteúdos programáticos que os alunos têm de estudar para cada disciplina. No entanto, se a teoria é base de todo o conhecimento, em áreas complexas como a programação ou a música, a prática não pode ser descorada. Aprender através da prática é fundamental para o melhoramento do desempenho e para o desenvolvimento das capacidades adquiridas através da teoria [GP05, dQ12].

Para aprender algo novo praticar não é suficiente. Enquanto se desenvolvem as capacidades nas mais diversas áreas, vão-se cometendo erros. E é com base nos erros que se vai aprendendo e desenvolvendo capacidades. Por isso, mais do que praticar, é necessário que alguém experiente e com conhecimentos esteja por perto, de forma a poder observar e indicar onde estão as falhas e de que forma podem ser corrigidas [dQ12]. Idealmente, a pessoa mais indicada para apoiar dessa forma os alunos seria o professor. No entanto, pelos motivos apresentados em cima, nem sempre isso é possível. A solução passa pela integração de mecanismos automáticos que possam dar esse apoio aos alunos.

As plataformas de *e-learning* permitem já aos alunos trocarem ideias, discutirem problemas e até mesmo esclarecerem dúvidas com o professor da disciplina. Da mesma

forma que permite aos professores partilharem com os alunos ficheiros e conteúdos que achem adequados, bem como avaliar os conteúdos adquiridos por eles de forma instantânea e automática [Zha03].

É com recurso a estas plataformas que o problema exposto pode ser resolvido. Por exemplo, em disciplinas de iniciação à programação é, frequentemente, utilizado um sistema de avaliação automático, que permite aos alunos saber imediatamente se a sua resposta está correta ou não. No entanto, o *feedback* fornecido nem sempre é o mais adequado para quem pretende aprender [TGP05, Jen02, SHP+06]. Isto porque para um aluno o mais relevante não é saber se a sua resposta está errada, mas sim indicação do erro de forma a que possa ser corrigido.

1.1 Motivação

A motivação para este projeto está associada ao *e-learning* e àquilo que pensamos ser uma lacuna no seu funcionamento: a avaliação de diagramas.

A avaliação automática de questionários de escolha múltipla ou de preenchimentos de espaços em branco veio proporcionar aos professores uma forma rápida e eficaz de testarem os conhecimentos dos seus alunos. O mesmo se aplica aos exercícios de programação que, numa avaliação manual são lidos pelo professor individualmente, o que é uma tarefa fastidiosa e sujeita a erros. A avaliação automática garante que todos esses erros serão ultrapassados e que os alunos serão avaliados uniformemente.

A avaliação automática pode ser utilizada não só para avaliação mas também como ferramenta de estudo. Desta forma proporcionam aos alunos uma ferramenta que lhe permite testar os seus conhecimentos e aprender com os seus erros. No entanto, tudo depende da qualidade do *feedback* que é dado. Quanto mais específicas e detalhadas forem as mensagens de erro relativas à resposta do aluno, maior probabilidade este terá de perceber onde errou e colmatar as suas falhas. Por outro lado, se apenas for indicado que a resposta está errada, o aluno terá muita mais dificuldade em ultrapassar o problema.

Neste trabalho o foco são os diagramas. Existem várias áreas onde são utilizados diagramas, por exemplo em: bases de dados, engenharia de *software* e circuitos elétricos, entre outros. Independentemente do tipo de diagrama a que se refira, quer a avaliação sumativa quer a avaliação formativa são tarefas custosas para os professores. Ambas seriam facilitadas por algo que permitisse efetuar uma avaliação e classificação

automática, uniforme e instantânea que permitisse aos alunos aprender e aperfeiçoar com a prática.

1.2 Objetivos

O objetivo principal deste projeto é criar um avaliador que recebe um diagrama com uma tentativa de resolução de um problema e o compara com um diagrama solução. Nas respostas erradas deve ser localizado o erro por forma a que o diagrama possa ser melhorado. Com base nesses erros é calculada a respetiva classificação.

Nas subsecções seguintes estão apresentadas a forma como são tratados os diagramas no contexto deste problema e uma breve descrição do funcionamento do sistema.

1.2.1 Diagramas e Grafos

Um diagrama é uma representação esquemática de informação. Essa representação tem associada a si elementos que possuem certas característica e um posicionamento no espaço. Abstraindo o *layout*, ou seja, a posição dos elementos, os diagramas podem ser representados por grafos. A abordagem que se pretende seguir para a avaliação dos diagramas é a comparação dos grafos que o representam. Assim é possível analisar o conteúdo do diagrama, sem dar relevância ao seu posicionamento ou formatação gráfica.

Pretende-se que o sistema comece por receber como *input* um diagrama sob o formato um ficheiro XML. Isto porque, o editor de diagramas utilizado tem de possuir a função de exportar sob esse formato. Em seguida, com um *parser* é necessário recolher desse ficheiro os dados relevantes ao problema, bem como do ficheiro que contém a solução. Depois de recolhidas as informações essenciais é necessário representar no sistema esses diagramas sob a forma de grafos. Daqui em diante passa a ser um problema de comparação de grafos e é aqui que pode ser afetado o tempo de execução (desde que foi feita a submissão até que é dado o *feedback*). É com base neste esquema de execução que se pretende integrar o avaliador num sistema de *e-learning*.

Para se compreender melhor é necessário ter presente a noção de grafo. Vulgarmente, a noção de grafo apresentada é a seguinte: $G = (N, E)$, ou seja, um grafo é um conjunto de nós e de arcos. Porém, neste trabalho pretende-se utilizar uma definição de grafo estendido, isto é, com mais características: $G = (N, N_t, N_p, E, E_t, E_p)$. Esta definição,

para além dos tradicionais conjuntos de nós e de arcos, contém as listas de tipos e de propriedades associadas a cada nó e arco. Estes tipos e propriedades representam características dos elementos dos diagramas.

1.2.2 Avaliação de Grafos

A Figura 1.1 representa de forma esquemática as interações entre os vários elementos fundamentais ao funcionamento do sistema que se pretende implementar.

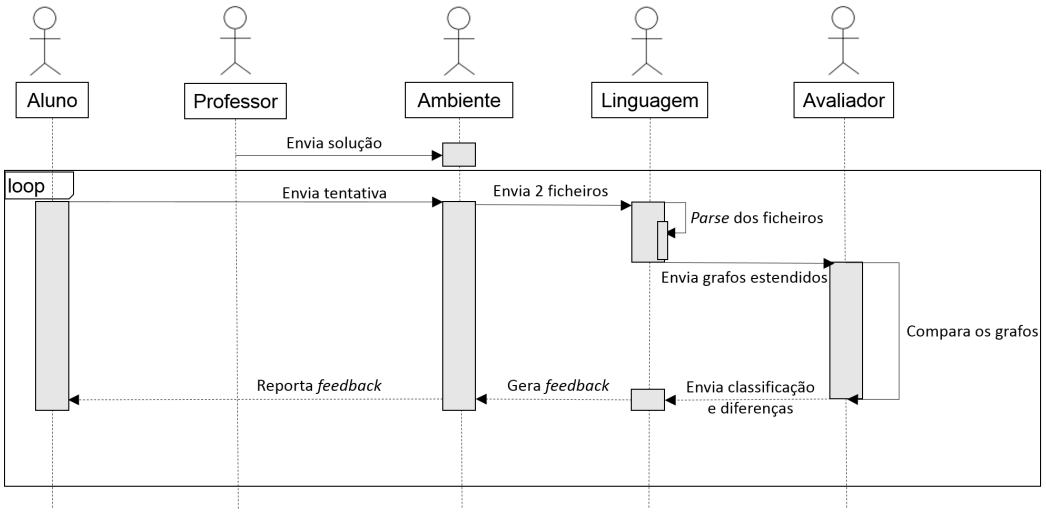

Figura 1.1: Esquema com breve descrição do funcionamento do sistema

Vejamos mais em concreto alguns dos passos descritos na Figura 1.1.

Os primeiros, quer do professor quer do aluno, prendem-se essencialmente com a modelação utilizando um software que permita a criação de diagramas e que permite a sua exportação sob o formato XML. Esses ficheiros terão como conteúdo uma série de elementos e atributos que identificam os objetos criados no modelo bem como as suas características. Os principais aspetos a ter em conta na análise do XML serão as representações dos tipos dos objetos, atributos e ligações e a relação existente entre eles. Após este passo, temos, no avaliador, a representação dos diagramas como grafos estendidos.

O passo seguinte é a comparação dos grafos. Este processo tem como objetivo descobrir qual o mapeamento dos nós solução em nós tentativa que permite minimizar o conjunto de diferenças e, consequentemente, maximizar a classificação. Para isso, é necessário descobrir que nó solução corresponde ao nó tentativa.

Neste momento estamos perante um problema de homomorfismo de grafos, isto é, de saber quais dos nós do grafo G correspondem aos nós do grafo G' e calcular a distância entre eles. Para isso, serão tidas em conta as características dos nós.

O homomorfismo é uma função de mapeamento dos vértices de um grafo nos vértices de um outro grafo que preserva os respetivos arcos. Seja f um homomorfismo do grafo $G = (V, E)$ no grafo $G' = (V', E')$, então temos que $f : V \rightarrow V'$ tal que $\forall\{u, v\} \in E\{f(u), f(v)\} \in E'$.

Para os grafos serem equivalentes f deve ser bijetiva, de forma a que a sua função inversa seja também um homomorfismo.

O problema associado à decisão da existência ou não de homomorfismo entre dois grafos é NP-completo. No entanto, no contexto deste trabalho os vértices e as arestas possuem características que permitirão utilizar heurísticas, de forma a tentar-se reduzir o número de testes até se chegar à solução final.

1.3 Estrutura da Dissertação

Esta dissertação é composta por seis capítulos, sendo que o primeiro deles é a Introdução.

No Capítulo 2 é apresentado o estado da arte, onde são resumidos os vários elementos da bibliografia utilizados no decorrer deste projeto. Os temas abordados são: o *e-learning*, os avaliadores automáticos de programação e escolha múltipla e os trabalhos relacionados no âmbito da avaliação de diagramas.

O Capítulo 3 apresenta uma descrição genérica dos métodos e das estruturas de dados que são utilizadas pelo avaliador. Por sua vez, o Capítulo 4 aborda a implementação dos métodos descritos anteriormente. Para além disso, é aproveitado para explicar, detalhadamente, cada uma das estruturas e métodos no contexto da avaliação de diagramas.

O Capítulo 5 sumariza os resultados obtidos durante a execução do projeto. Inicialmente são apresentados os motivos que levaram à escolha do editor de diagramas e à rejeição das outras alternativas. Posteriormente, são analisados os resultados decorrentes das experiências realizadas com dados sintéticos e com alunos e problemas reais.

O último capítulo sintetiza as conclusões decorrentes do trabalho realizado. Para além disso, aborda o trabalho futuro nesta linha de investigação.

Capítulo 2

Estado da Arte

2.1 *E-learning*

"E-learning enables unique forms of education that combine the existing paradigms of on-campus and distance education. "

[Nic08]

O ensino e a aprendizagem são dois dos aspetos mais importantes na construção e evolução de uma sociedade. Só com o estudo e aplicação de conhecimentos é possível alterar e melhorar todos os aspetos da atividade humana.

Um dos resultados mais notórios dessa evolução pode ser visto ao nível da tecnologia. O tempo de estudo dedicado às redes de comunicação e de computadores permitiu que estas se desenvolvessem de forma extraordinária, ao ponto de hoje em dia serem um instrumento essencial em qualquer empresa, universidade ou hospital, entre muitos outros.

A evolução vivida ao nível das comunicações tem levado a uma alteração, no setor da educação, na forma em que é feito o ensino. Isto porque tem originado um desvio na mentalidade das pessoas. Se até aqui, o método tradicional era centrado no professor e naquilo que era apresentado nas aulas presenciais, atualmente acredita-se mais numa filosofia de ensino centrado no aluno. Isto tem permitido que os alunos se tornem mais autónomos no que diz respeito aos temas e às alturas em que pretendem estudar. Podemos estar perante a passagem do ensino por exposição para a aprendizagem pela prática [WG05].

Esta revolução tem sido bastante ajudada pelas plataformas de *e-learning* que, em vez de substituírem por completo o método de ensino tradicional, vem completá-lo com mecanismos e serviços de apoio ao estudo dos alunos, independentemente do local onde se encontrar e das horas a que pretendem estudar [Zha03].

2.1.1 Recursos

As plataformas de e-learning, também denominadas como *Learning Management System* (LMS), fornecem aos utilizadores um ambiente onde é possível disponibilizar e gerir recursos que são partilhados com um certo grupo de pessoas que estão ligadas por um tema ou curso em comum.

Existem LMS *open source* e proprietários. No entanto, apenas os livres, como por exemplo o *Moodle*, para além de disponibilizarem um vasto leque de recursos e serviços, disponibilizam também uma *framework* que pode ser alterada e que permite a cada instituição ter o seu próprio sistema de *e-learning* personalizado, apesar de ter como base algo comum a várias outras instituições [CCP04].

Os principais serviços disponibilizados, e utilizados em âmbito académico, por este tipo de plataforma são [Nic08]:

- Criar e gerir o conteúdo de um curso
- Armazenar informação relativa ao percurso académico de um aluno
- Fórum de discussão entre alunos (e com o professor)
- Envio de mensagens privada entre os vários membros
- Submissão de trabalhos e armazenamento associado ao respetivo professor do curso
- Ferramentas de avaliação automática de exercícios (escolha múltipla, preenchimento de espaços em branco, programação, etc.)

Nem todas as plataformas possuem todos estes serviços, no entanto, existem também algumas delas que possuem outros tipos de serviço não identificado acima.

Para além disso existe uma distinção entre os serviços síncronos e os assíncronos. Essa diferença assenta na forma como é feita a comunicação. Os serviços síncronos exigem

que os intervenientes utilizem a plataforma em simultâneo. Um exemplo deste tipo de serviço são as vídeo-aulas que requerem uma transmissão em direto de uma aula em que, se o aluno quiser assistir, terá de se encontrar *online* nesse preciso momento. No entanto, existem professores que optam por gravar a sua aula mas, em vez de uma transmissão em direto, fazem o *upload* do vídeo na plataforma, permitindo aos alunos escolher quando querem ver a aula. Este serviço é então assíncrono, visto que a altura em que é visto o vídeo não tem de ser o instante em que ele é colocado.

Todos estes serviços têm vindo a fornecer uma nova experiência a alunos e professores no desempenhar das suas funções.

2.1.2 Vantagens

O e-learning tem-se mostrado uma boa solução e um bom complemento para as necessidades dos alunos e professores devido às inúmeras vantagens que lhe estão associadas. No entanto, como qualquer tecnologia e plataforma acarreta alguns inconvenientes.

As principais vantagens apontadas a esta tecnologia são [Zha03]:

- **Flexibilidade de tempo e local:**
  Os alunos e professores podem aceder quando quiserem e em qualquer local, com ligação à Internet, aos conteúdos disponibilizados na plataforma.

- **Redução de custos:**
  Esta vantagem está associada apenas aos cursos inteiramente *online*, visto que permitem que os alunos se inscrevam, assistam às aulas e acedam aos recursos sem ter a necessidade de se deslocarem. Desta forma estão a poupar em tempo de viagem e em custos associados ao transporte.

- **Aprendizagem pessoal:**
  Um aluno pode aceder apenas aos conteúdos que menos conhece e mais precisa de estudar. Ao contrário de uma aula de esclarecimento de dúvidas, onde todos vão expondo as suas dúvidas, independentemente de quem sabe ou não esses conteúdos, o e-learning dá a possibilidade de cada indivíduo rever e exercitar aquilo que acha mais adequado para si.

- **Aprendizagem cooperativa:**
  Esta vantagem não invalida a anterior, uma vez que esta refere-se ao facto de, no mesmo espaço virtual, se encontrarem pessoas que dominam um certo tema e

outras que não e, desta forma, ser possível através dos fóruns e *chats* ajudarem-se mutuamente.

- **Maior disponibilidade do professor:**
  Esta vantagem está assente em dois pontos específicos. O primeiro é relativo à dificuldade que existe em encontrar alguns professores disponíveis pessoalmente, visto que se encontram, muitas vezes, com um elevado número de tarefas para executar. A possibilidade de a qualquer momento aceder à plataforma e responder a algumas dúvidas, torna o contacto entre professor e aluno mais frequente. O segundo aspeto está associado à avaliação automática de exercícios que estas plataformas proporcionam e que libertam os professores dessa tarefa, dando-lhe maior disponibilidade.

Relativamente às desvantagens, a principal prende-se com a falta de contacto direto entre o professor e os alunos, visto que a existência de material disponível (sejam *slides*, vídeo-aulas, etc.) ajuda mas nem sempre é suficiente para esclarecer algumas dúvidas.

2.2 Avaliadores Automáticos

A avaliação é um processo que exige bastante trabalho por parte de quem a prepara. Começando com a escolha das questões a fazer, passando pela correção e classificação das respostas, as deteções de plágio até à divulgação dos resultados, estas tarefas do professor reduzem o tempo disponível para esclarecer os alunos. De forma a simplificar o processo de avaliação, existem agora mecanismos de avaliação automática (*e-assessment*) [Jay10].

Os avaliadores automáticos são uma ajuda no ensino. Através das plataformas disponíveis, estes permitem que os alunos possam resolver exercícios de certas matérias e obter um *feedback* imediato relativo às suas respostas.

Os avaliadores automáticos são utilizados, por exemplo, nos exercícios de programação. No entanto, existem também formas de avaliar testes com respostas de escolha múltipla ou completar espaços em branco [dQ12].

Este tipo de mecanismos pode ser utilizado não só na fase de avaliação mas em exercícios de consolidação de conhecimentos durante o curso ou, até mesmo, em exercícios de diagnóstico, antes de começarem o curso [Jay10].

2.2.1 Questionários

Existem vários sistemas que permitem a utilização de ferramentas para a correção automática de questionários. Na UP, por exemplo, um dos sistemas utilizados é o Moodle.

Os principais sistemas baseiam-se na especificação *Question and Test Interoperability* (QTI). As especificações QTI definem regras para a representação do conteúdo a avaliar bem como dos resultados obtidos. Para além disso fixa a forma como devem ser efetuadas as trocas entre o sistema e o autor do teste [dQ12].

Esta representação é um modelo de dados onde se encontra definida a estrutura das perguntas e um conjunto pré-definido de respostas, as respostas dadas, o *feedback* dado pelo sistema e o resultado final. É baseado numa norma XML que define as trocas de mensagens entre os vários participantes na avaliação.

2.2.2 Programação

Nos exercícios de programação a avaliação automática difere um pouco dos questionários. Como não existe uma única resposta correta que possa ser indicada ao sistema, não pode ser utilizado o QTI.

A forma mais comum de se avaliar é com recurso a casos de teste. Dois exemplos de sistemas que utilizam este método são: o *Mooshak* e o *DOMjudge* [dQ12].

O *Mooshak* foi criado com o intuito de servir como avaliador para concursos de programação baseados nas regras da *International Collegiate Programming Contest* (ICPC). No entanto, tem vindo a ser, cada vez mais, utilizado em ambiente universitário como recurso de estudo e avaliação no ensino da programação [Pac10].

A avaliação no *Mooshak* é feita por comparação de *outputs*. Para isso, o aluno ou equipa devem submeter o código fonte de resposta ao problema que lhe foi fornecido. Esse ficheiro será alvo de dois tipos de análise distintas: estática e dinâmica [LS03].

Em primeiro lugar é feita a análise estática, onde se realiza a verificação do ficheiro no que diz respeito ao seu tamanho. Se o tamanho não exceder o limite imposto, o sistema vai tentar compilar o código. Caso alguma destas situações não se verifique, a solução não é classificada e é dada essa informação ao aluno. Caso passe ambos os testes, o ficheiro tem de passar a análise dinâmica.

É na fase da análise dinâmica que a solução vai ser efetivamente testada. Os testes são executados depois do código estar compilado, com uma série de casos de teste. Estes casos de teste são compostos por ficheiros com casos de *input* e ficheiro com as respetivas soluções. Para cada execução do programa com um input é comparado o *output* com o esperado. Consoante o número de testes que o programa passe será atribuída uma classificação diferente.

Para além de se obter como *feedback* "X Wrong Answer" (sendo X a classificação atribuída à solução) ou "100 Accepted" existem outros tipos de resposta possíveis, por exemplo:

- Presentation Error - A solução está correta mas não está apresentada da forma pedida
- Time-limit Exceeded - A execução de programa excedeu o tempo limite (poderá conter um ciclo infinito ou um algoritmo de complexidade temporal elevada)
- Run-time Error - Ocorreu um erro durante a execução de algum caso de teste

O *DOMjudge* funciona de forma semelhante ao *Mooshak*. Este sistema é utilizado nas competições regionais e internacionais de programação da ICPC. Neste tipo de concursos, as equipas estão num computador onde resolvem os problemas propostos e submetem o seu código que será avaliado num computador de um júri [DOM].

Após a submissão do código, o ficheiro fica numa fila de espera para ser compilado. Assim que exista um computador do júri livre, esse ficheiro é automaticamente compilado.

Em seguida, acontece o mesmo que no exemplo anterior. O programa é executado com os casos de teste e são comparados os resultados obtidos com os esperados. No *DOMjudge* podemos obter o seguinte tipo de repostas:

- Accepted - Resposta correta
- Wrong-Answer - Resposta errada
- Presentation-Error - Resposta possivelmente correta mas apresentada de forma incorreta
- Compiler-Error - O código não foi compilado
- Run-Error - Ocorreu um erro durante a execução

- Timelimit - A execução do programa excedeu o tempo limite
- No-Output - O programa terminou sem a apresentação de resultado

Para evitar destabilizar o sistema, são impostas limitações no que diz respeito ao tempo de compilação, ao tamanho do ficheiro, à quantidade de memória utilizada na execução do programa e no número de *threads* que são criadas.

2.3 Trabalhos Relacionados

A área da avaliação automática de diagramas tem vindo a ser alvo de estudo. Existem já algumas publicações que abordam essa temática. Nesta secção serão descritos alguns desses trabalhos.

A pesquisa feita levou-nos a concluir que nenhum dos trabalhos revistos apresenta uma solução para o problema tão ampla e abrangente como a que nos propomos apresentar. Contudo, existem, também, algumas semelhanças de ideias. As várias características comuns e distintivas serão apresentadas nas secções seguintes e sintetizadas no final.

2.3.1 Avaliador faseado de diagramas de classes (UML)

Uma das publicações analisadas apresenta uma solução para a avaliação de diagramas de classes UML. O artigo apresentado por Ali et al. [ASI07] descreve o funcionamento de um mecanismo de avaliação que denominaram de *UML Class Diagram Acessor* (UCDA).

O UCDA, tal como o que pretendemos apresentar com este projeto, espera receber dois ficheiros contendo uma solução e uma tentativa de resposta. Após posterior avaliação, é enviado uma mensagem com sugestões relativas aos erros encontrados.

Contudo, um aspeto diferenciador é que este sistema pretende executar uma avaliação faseada. Isto é, durante a sua execução existem três momentos distintos e independentes de avaliação. Os três módulos de avaliação são sequenciais e são os seguintes:

1. Análise estrutural das classes
2. Processo de verificação

3. Verificação de linguagem

A primeira fase consiste numa verificação estrutural das classes. A análise estrutural tem como objetivo verificar os seguintes pontos:

- O número de classes é o esperado
- O número de atributos na classe <C>[1] é o esperado"
- O número de operações na classe <C>[1] é o esperado
- O tipo do atributo <A>[2] na classe <C>[1] é o esperado
- O tipo do parâmetro da operação <O>[3] na classe <C>[1] é o esperado

Caso qualquer uma das condições acima seja violada é enviado um *feedback* relativo ao problema detetado e é parada a execução nesse instante, impossibilitando a passagem para a segunda fase de avaliação. Apenas no caso de todas as condições se verificarem é que é chamada à execução o processo de verificação.

O processo de verificação consiste na análise das ligações entre as classes. Essas ligações são verificadas da seguinte forma:

- O número de relacionamentos é o esperado
- O número de relacionamentos do tipo <T> é o esperado
- O relacionamento entre $<C_1>$[4] e $<C_2>$[4] está correto

Inicialmente é verificado o número total de relacionamentos, seguido do número de relacionamentos por cada tipo. Apenas quando estas condições se verificam é que são comparadas as extremidades das ligações. Em caso de falhar alguma destas condições, mais uma vez é interrompida a execução e enviada uma mensagem com o erro. Em caso positivo, avança-se para o estado final.

O último estado de avaliação prende-se com uma análise linguística e de nomenclatura dos elementos. Apesar de não obrigar a que os nomes da tentativa sejam iguais aos da solução, estes têm de verificar as seguintes restrições:

[1] <C> representa o nome da classe
[2] <A> representa o nome do atributo
[3] <O> representa o nome da operação
[4] $<C_1>$ e $<C_2>$ representam nomes de classes

- O nome da operação <O>[3] da classe <C>[4] é um verbo
- O nome do atributo <A>[2] da classe <C>[1] é um nome
- O nome da classe <C>[1] é um nome

Após passar as três fases de avaliação, a tentativa é considerada correta.

2.3.2 Avaliador de diagramas de classes (UML) com nomes fixos

Uma outra sugestão para um avaliador de exercícios de classes UML é apresentado por Soler et al. [SBP+10]

Esta investigação utiliza potencialidades de algum trabalho prévio desenvolvido no mesmo departamento relativo a diagramas ER, nomeadamente, o seu editor de diagramas.

A abordagem proposta é genericamente semelhante: é proposto um exercício que, após ser resolvido, é submetido no sistema e avaliado. Posteriormente, é enviado um *feedback* decorrente dessa mesma avaliação. Contudo, uma das diferenças apresentadas prende-se com o diagrama solução. Estes autores propõe que, devido à possibilidade de existir mais do que uma solução correta para o mesmo problema, a tentativa do aluno deve ser comparada com um conjunto de soluções corretas. As mensagens de erro serão relativas à solução que apresente um menor número de diferenças.

A grande diferença nesta abordagem está na forma como é feito o mapeamento entre os nós da solução e os nós da tentativa. A chave deste processo é a exigência de se manter o nome dos atributos exatamente igual ao que surge no enunciado do problema. Este é o único requisito exigido para que este avaliador funcione corretamente. Porém, tem um impacto enorme no seu desempenho, visto que torna todo o processo de encontrar correspondência entre nós muito simples. Para isso, basta procurar, de um lado e do outro, os nós que têm os atributos com os nomes requeridos e atribuir-lhes correspondência.

Relativamente ao *feedback* produzido é focado essencialmente no número de elementos (classes, associações, etc.) e no tipo desses mesmos elementos e das suas propriedades.

2.3.3 Avaliador de diagramas de casos de uso (UML)

Vachharajani e Pareek [VP14] apresentaram a ideia de um avaliador automático especializado em diagramas de casos de uso.

A primeira abordagem apresentada contempla a utilização de um editor de diagramas, porém para uma abordagem posterior, é pretendido implementar o seu próprio editor de diagramas, com as características desejadas.

O avaliador tem duas componentes a serem executadas paralelamente: a componente de avaliação de *labels* e a componente da avaliação da estrutura.

No que aos nomes diz respeito, e contrariamente à ideia demonstrada em secções anteriores, esta abordagem não obriga a utilização dos mesmos nomes descritos no enunciado dos problemas. Em vez disso, pretendem fazer um estudo detalhado da semântica das palavras utilizadas. Porém, os desafios encontrados e que se propuseram resolver foram os seguintes:

- A utilização de caracteres especiais (por exemplo: *underscore*
- Os erros ortográficos
- A concatenação de palavras (por exemplo: "ReportarErro" em vez de "Reportar Erro")
- A utilização de sinónimos e abreviaturas (por exemplo: "Ler msg" em vez de "Ler mensagem" ou "Verificar utilizador" em vez de "Autenticar utilizador"
- Diferente número de palavras (por exemplo: "Registar" em vez de "Registar Utilizador")
- Problemas não diretamente relacionados com linguística (por exemplo: "Utilizador", "Pessoa" ou "Cliente" em vez de "Passageiro")

No que diz respeito ao avaliador estrutural, a ideia é criar um agente que trate do mapeamento entre os vários elementos do modelo da solução e da tentativa do aluno. Aqui os desafios apresentados são semelhantes às outras abordagens, visto que têm em conta o número de agentes e casos de uso, a direção das ligações, entre outras questões relevantes para a estrutura do grafo. No entanto, apresentam uma abordagem que difere das anteriores: a possibilidade de um aluno juntar ou separar informações nos casos de uso. Por exemplo, se um agente tem a capacidade de calcular o lucro e a perda

de uma empresa, essa informação pode ser representada conjuntamente ("Calcular ganhos e perdas") ou separadamente ("Calcular ganhos" e "Calcular perdas"). Esta abordagem sugere, e bem, que numa situação destas nenhuma das alternativas pode ser considerada errada, pois a informação representada é a mesma.

No entanto, esta primeira fase não contempla o envio de *feedback* ao aluno estando isso previsto para uma fase posterior.

2.3.4 Avaliador de DFA

Automata Diagram Assessment Tool (ADAT) [SM08] é um sistema construído para avaliar autómatos, neste caso autómatos finitos determinísticos (DFA). Contrariamente aos exemplos anteriormente apresentados, os autómatos são diagramas que representam informação dinâmica. Isto é, se por um lado a informação contida nos diagramas UML ou ER não depende de nenhuma execução, este tipo de diagramas tem de ser avaliado tendo em conta os resultados e/ou efeitos que a sua execução provoca.

Desta forma, a proposta apresentada contempla dois tipos de avaliação: uma avaliação estática e uma avaliação dinâmica.

A avaliação estática consiste na análise estrutural do diagrama. Isto é, na comparação do número global de estados, do número de estado iniciais e finais e no número de transições entre o diagrama solução e a tentativa.

Posteriormente é realizada a avaliação dinâmica. Esta consiste na verificação do comportamento do DFA perante um conjunto de *strings*. Para isso, é fornecido ao sistema um conjunto de palavras que devem ser aceites pelo autómato. Caso alguma delas não seja reconhecida, pode ser dado como *feedback* o que deveria ter sido aceite e não foi. Analogamente, em sentido inverso, existe um conjunto de *strings* que devem ser rejeitadas pelo autómato. Caso alguma delas sejam reconhecida, deve ser reportada.

O funcionamento descrito nesta abordagem difere dos restantes, pois apresenta um instante em que é avaliado o comportamento do diagrama (isto é, o resultado da sua execução) e porque para além de receber como *input* o diagrama solução e tentativa tem de receber, também, o conjunto de casos de teste que pretende verificar. Em comum tem a fase da avaliação estrutural e o *feedback* relativo a isso.

2.3.5 Avaliador de diagramas genérico

Thomas et al. [TWS12] apresentam um sistema de avaliação próximo daquilo que se pretende com este projeto. O primeiro ponto em comum é que este artigo expõe uma abordagem que permite a avaliação de diferentes tipos de diagramas.

A segunda característica comum é a forma como é feito o mapeamento entre os nós e arcos da solução e os da tentativa. Para cada par de nós e arcos da solução e tentativa é calculada uma razão de proximidade que apresenta valores entre 0 e 1. Depois, pretende escolher-se os pares cuja soma das razões seja máximas.

Porém, a diferença nesta abordagem para o que é desejável neste projeto é que, qualquer razão muito próximo de 1 é assumido como sendo uma correspondência perfeita, não se mapeando esses elementos em nenhum outros, e qualquer razão próxima de 0 é considerada impossível, não se considerando esse mapeamento como possibilidade. Ou seja, é assumido que algo estará correto mas sem certezas.

2.3.6 Conclusões

Nesta secção foram apresentadas algumas das principais abordagens propostas no âmbito da comparação automática de diagramas. A maioria dos trabalhos já apresentados nesta área focam-se numa linguagem específica. Este é um dos pontos diferenciadores para o nosso projeto que pretende ser o genérico e abrangente de forma a ser possível comparar diagramas de diferentes tipos com recurso ao mesmo algoritmo.

Como se pretende fazer uma comparação apenas estrutural, os nomes dos objetos não serão importantes na nossa avaliação. No entanto, a utilização dos nomes é uma das formas mais utilizadas para identificar correspondências entre nós do diagrama solução e nós do diagrama tentativa. Este método é o mais eficiente visto que procura pelo nome e emparelha os que tiverem nome igual. No entanto, é bastante redutor, uma vez que no tipo de exercícios que pretendemos avaliar, por norma as etiquetas utilizadas para identificar os nós não têm muita importância e podem assumir diversos valores com o mesmo sentido.

Uma das propostas demonstra como fazer uma avaliação faseada. É uma abordagem interessante, porém não permite ir enviado ao aluno a nota de cada comparação, visto que, não são comparadas todas as características dos diagramas em simultâneo. É desejável que, a cada submissão, seja realizada uma comparação total, para que seja possível em todas elas se encontrar um conjunto de diferenças completo e calcular-se

uma nota decorrente dele.

O último trabalho apresentado nesta Secção é o que, de todos os analisados, tem mais semelhanças como o que se pretende implementar. Desde logo porque é uma proposta para avaliar diagramas independentemente do seu tipo. Em seguida, porque pretende fazer uma análise estrutural dos diagramas, podendo, assim, abstrair-se dos nomes. A principal diferença está na forma como se encontra o melhor mapeamento entre os nós solução e os nós tentativas. Para um par de nós é considerado que um mapeamento é correto se o valor da sua comparação for elevado, e considerado incorreto se a sua comparação retornar um valor baixo, excluindo-se essa possibilidade. Contudo, no nosso caso, o que se pretende é considerar essas situações para computar uma ordem para serem testados os mapeamentos. Excluindo-se apenas uma possibilidade quando se tem completa certeza de que essa alternativa seria pior do que outras.

Capítulo 3

Avaliação de Diagramas

Este capítulo apresenta a descrição de todos os aspetos relevantes na implementação. Começa por apresentar a descrição de como fazer a comparação dos grafos, e em seguida descreve as estruturas de dados utilizadas, a forma como é feita a avaliação e ainda apresenta o funcionamento do *parser* e do afinador de parâmetros.

3.1 Comparação de Grafos

A avaliação proposta nesta dissertação pretende ser genérica e adaptável a qualquer tipo de diagrama.

Para tanto é seguida a seguinte abordagem: 1) o sistema recebe dois ficheiros (um com a solução e outro com a tentativa) bem como o tipo de diagrama que vai avaliar; 2) os ficheiros são convertidos em grafos de acordo com o seu tipo; 3) os grafos são comparados sendo esta comparação parametrizada em função do seu tipo e é retornado uma nota e um conjunto de diferenças entre os dois grafos; 4) desse conjunto de diferenças e, com base no tipo de diagramas avaliados, é processado o *feedback* a apresentar ao aluno. O esquema da Figura 3.1 exemplifica este funcionamento.

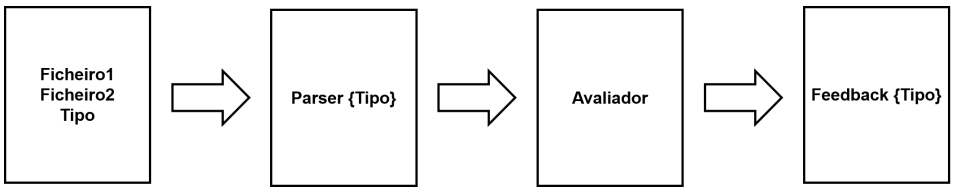

Figura 3.1: Esquema do funcionamento do programa

3.1.1 Definição dos Grafos

Para abordarmos este problema, definiu-se uma estrutura de forma a representar um grafo estendido. Esse grafo pode ser representado da seguinte forma:

$$G = (N, V, t_n, p_n, t_v, p_v)$$

sendo que:

N: conjunto de nós do diagrama $n \in N$

V: conjunto de ligações do diagrama $(n_i, n_j) \in V$

t_n**:** função $t_n : N \to T_n$ onde T_n é o conjunto de todos os tipos de nós possíveis

p_n**:** conjunto de propriedades dos nós

t_v**:** função $t_v : V \to T_v$ onde T_n é o conjunto de todos os tipos de ligações possíveis

p_v**:** conjunto de propriedades das ligações

Num grafo com um conjunto de nós e outro de arcos, cada elemento num desses conjuntos tem associado a si um determinado tipo e um conjunto de propriedades que serão essenciais na comparação entre dois grafos.

3.1.2 Distância de Grafos

A distância entre dois grafos G e G' representa o quão semelhantes ou quão diferentes esses dois grafos são. Esta semelhança é medida através da comparação dos seus nós e arcos.

Para comparar um nó $n \in G$ com $n' \in G'$, deve ser tido em conta se o tipo é o mesmo ou não, quantas das propriedades são equivalentes e qual a diferença nos seus graus. O grau de um nó indica o número de ligações que têm esse nó como extremidade e pode estar dividido em grau de entrada (o nó é a extremidade final da ligação) e grau de saída (o nó é a extremidade inicial da ligação).

O caso dos arcos é em tudo semelhante. No entanto, o principal fator de comparação de dois arcos são as suas extremidades. Isto é, se o arco $v \in G$ tem como origem e

como destino os mesmos que o arco $v' \in G'$, então serão comparados os seus tipos e propriedades. Caso contrário, é assumido que não existe qualquer semelhança entre essas duas ligações.

A proximidade dos nós (ou arcos) é igual à soma das proximidades de todos os nós (ou arcos). Após estas comparações individualizadas é calculado uma nota de proximidade global do grafo:

$$proximidade = proximidade_{nos} + proximidade_{arcos}$$

Este valor de proximidade está parametrizado para que o seu valor esteja no intervalo $[0, 100]$, sendo que 0 indica que os dois grafos são completamente diferentes e que 100 representa uma igualdade entre as duas representações.

3.1.3 Parâmetros de Avaliação

Como descrito na Secção 3.1.2, para se obter um valor no intervalo pretendido é necessário utilizar alguns parâmetros que normalizem esse valor.

Comecemos por ver como são calculadas as comparações entre os elementos (sejam nós ou arcos). O primeiro aspeto a reter é que nem todos os nós e arcos contribuirão para a classificação final do mesmo modo. Vejamos porquê.

Tenhamos presente que num diagrama representado por um grafo estendido, os vários elementos variam a sua composição. Isto é, têm tipos e números de propriedades diferentes. Quantas mais propriedades tiver um nó, maior impacto esse nó terá na classificação. Não seria correto assumir que um nó que não possui nenhuma propriedade contribui de igual forma na nota da avaliação como um nó com várias propriedades. Então, quantas mais propriedades tiver um nó, maior impacto esse nó terá na classificação.

Para isso definiu-se que qualquer elemento, seja um nó ou um arco, tem um valor associado. Esse valor resulta da atribuição de parâmetros e é utilizado no cálculo da comparação dos elementos. A fórmula de cálculo do valor é a seguinte:

$$valor = peso\_do\_tipo + peso\_da\_propriedade \times numero\_de\_propriedades$$

No caso de o elemento em questão ser um nó, ao valor é ainda acrescentado:

$$peso\_do\_grau \times grau\_do\_no.$$

De notar que todos os pesos utilizados (tipo, propriedade e grau) são constantes e iguais para todos os elementos de um grafo e dependem do tipo de diagrama que o grafo representa. O peso referente ao tipo é atribuído apenas pela existência de um tipo e terá apenas impacto na comparação dos nós.

Não seria correto assumir que estes três pesos como iguais pois o impacto de faltar um tipo (ou ser diferente do esperado) será maior do que a falta de uma propriedade. Do mesmo modo, na comparação de dois nós que possuem o mesmo tipo e todas as propriedades são coincidentes, o peso do grau não deve influenciar muito negativamente a avaliação. Então temos que:

$$peso\_do\_tipo > peso\_da\_propriedade > peso\_do\_grau.$$

Para além disso não se pretende atribuir o mesmo peso à classificação dos nós e dos arcos. Este ponto surge devido à importância que ambos os elementos trazem na representação de um sistema. Isto é, a falta de um nó num diagrama será, por norma, algo muito mais crítico do que a falta de uma ligação. Por isso, um dos parâmetros utilizados é que: $peso_{nos} > peso_{arcos}$.

Temos então que que a nota do grafo é calculada por:

$$nota = peso_{nos} \times \sum valor(no) + peso_{arcos} \times \sum valor(arco)$$

Todas estas relações entre os vários parâmetros foram deduzidas e posteriormente confirmadas através de um afinador de parâmetros, descrito na Secção 3.5

3.1.4 Emparelhamento de Nós

Este projeto tem com objetivo encontrar um conjunto de diferenças entre dois grafos. No entanto, esse problema é complexo visto exigir uma correspondência entre os nós de ambos os grafos. Ao fazer essa correspondência está a criar-se ao mesmo tempo uma correspondência entre os seus arcos. Assumindo que n_1, n_2 são nós do grafo G e que são mapeados, respetivamente, nos nós n'_1, n'_2 do grafo G', temos que com esta condição, os arcos de G que tenham como extremidades os nós n_1, n_2 serão mapeados nos arcos de G' que têm como extremos os nós n'_1, n'_2.

Existem várias formas de fazer este mapeamento de nós. A forma mais simples é fazer corresponder a lista de nós do grafo solução com todas as permutações possíveis da lista de nós do grafo tentativa e compará-los. O que obtivesse a classificação mais elevada seria considerado o mapeamento correto. Esta hipótese garante que se encontre o melhor mapeamento possível entre os dois grafos, já que todos os possíveis são testados. Contudo, o facto de exigir que sejam testadas todas as permutações, faz com que, para grafos a partir de um determinado número de nós (inferior a 10), seja incomportável a utilização deste método devido ao seu excessivo tempo de execução. Esta afirmação pode ser confirmada na Secção 4.2.1.

O objetivo é então a criação de um algoritmo que permita analisar as diferenças de grafos de dimensão elevada (a rondar os 30 nós). Para isso criou-se um algoritmo que recorre a heurísticas que permitem análise dos mapeamentos gerados numa ordem que é mais favorável. Para além disso, o algoritmo permite evitar que sejam analisadas todas as permutações, pois consegue identificar um ponto a partir do qual será impossível melhorar o resultado. Os detalhes relativos a esta implementação encontram-se nas secções seguintes.

3.2 Estruturas de Dados

O tipo de dados principal deste projeto é o grafo estendido. São instâncias deste tipo que são comparadas na avaliação. Na sua definição são utilizados outros tipos que representam nós, arcos e propriedades. Para além disso, existem tipo de dados para representarem as diferenças entre os grafos.

Nas subsecções seguintes estão descritas estas estruturas.

3.2.1 Grafo Estendido

A estrutura do grafo é descrita pelo diagrama de classes da Figura 3.2. Como se pode observar, um grafo é constituído apenas por duas listas: uma de nós e uma de arcos. Contudo, esses nós e arcos contém informações que serão fundamentais na comparação de grafos.

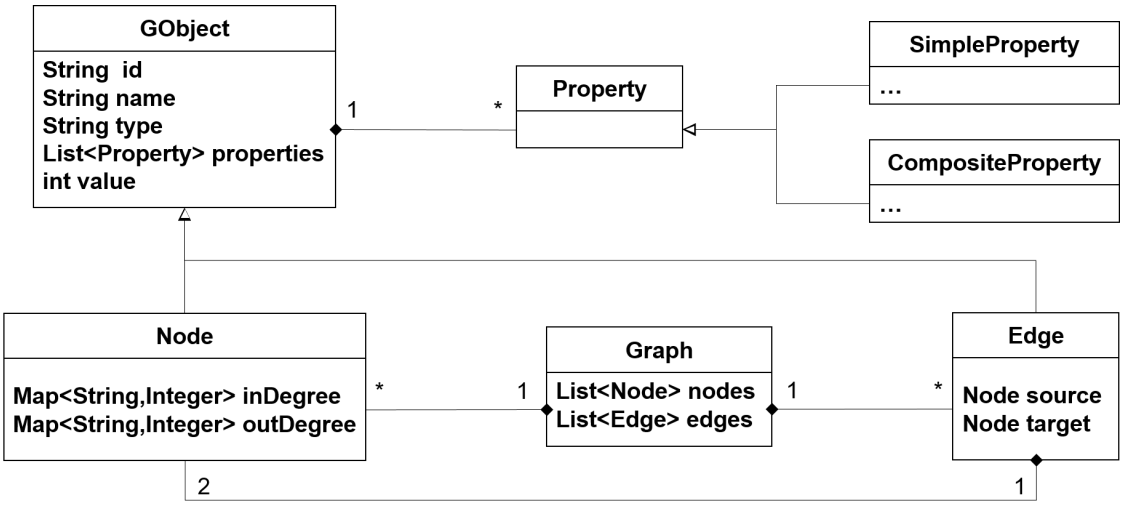

Figura 3.2: Representação esquemática das estruturas de dados

3.2.2 Nós e Arcos

Os nós (*Node*) e os arcos (*Edge*) são os tipos de estruturas mais importantes pois é neles que estão contidos dados suscetíveis a comparações. Isto porque a comparação entre dois grafos divide-se na comparação dos seus nós e dos seus arcos.

Apesar de nós e arcos representarem estruturas diferentes, possuem muitas características em comum. Por isso, optou-se pela criação de um tipo de dados mais abstrato que representa um objeto de um grafo (*GObject*) e que agrupa todas essas características.

Um *GObject* contém: uma *string* que identifica o objeto (`id`), uma *string* com o seu nome (`name`), uma lista de propriedades (`properties`) e um valor calculado com base no seu tipo e propriedades (`value`).

Para além das características comuns, os nós e os arcos têm elementos que os distinguem. A estes elementos, nos arcos, são acrescentados dois nós: um nó referente à origem da ligação e um no referente ao destino da ligação. É através desta caraterística que se pode afirmar que os grafos utilizados são dirigidos.

Nos nós, os elementos diferenciadores estão relacionados com os seus graus de entrada e saída, ou seja, o número de ligações que têm como origem e o número de ligações que têm como destino o nó referente, respetivamente. Esses valores estão agrupados por tipo. Daí serem armazenados numa estrutura do tipo *Map<String, Integer>*. Nesta estrutura são guardadas como *String* o tipo da ligação e como inteiro o número de ligações desse mesmo tipo.

Na Figura 3.2 também estão exemplificadas estas estruturas.

3.2.3 Propriedades

Outra estrutura relevante na comparação dos constituintes de dois grafos são as propriedades. As propriedades são elementos que caracterizam um objeto no entanto não apresentam todas a mesma estrutura. Umas são mais complexas do que outras. E isso levou à criação de dois tipos de propriedades: simples e compostas.

Para explicar melhor a diferença entre elas é vantajoso tomar como exemplo um diagrama UML de classes. Existem vários tipos de propriedades relativas a este tipo de diagramas. Num arco pode ter-se a cardinalidade, que se representa na extremidade das ligações e que é apresentado de forma muito simples como uma propriedade: "cardinalidade_origem = 1", "cardinalidade_destino = *". Este tipo de propriedade é considerado uma propriedade simples, pois existe apenas uma relação direta entre o nome e o seu valor. Já uma classe com atributos e operações é mais complexo porque um mesmo atributo ou operação pode conter várias caraterísticas. Tomemos como exemplo uma classe Pessoa que tem como um atributo "nome". Aqui estamos perante uma propriedade composta porque o atributo tem um tipo (*String*, *int*, etc.), visibilidade (*public*, *private*, etc.), e valor (nome = "João", nome = "Manuel", etc.), entre outros.

Tendo por base esta diversidade optou-se por dois tipos de estrutura diferente para representar as propriedades. A propriedade simples exige uma implicação direta entre o nome da propriedade e o seu valor. A propriedade composta agrupa numa só propriedade um conjunto de subpropriedades referentes à mesma.

Na Figura 3.3 vemos dois objetos com duas propriedades cada. No entanto, o arco possui duas propriedades simples e o nó apresenta duas propriedades compostas.

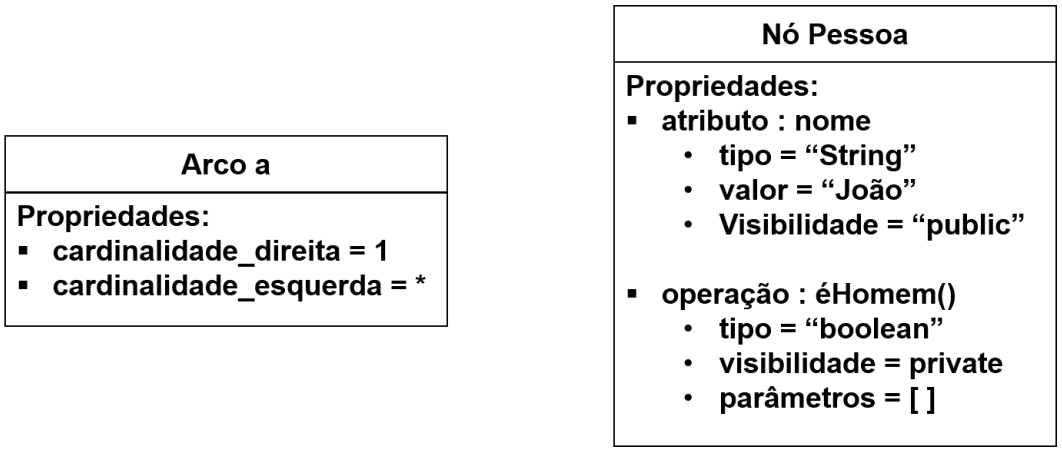

Figura 3.3: Propriedades simples e compostas

3.2.4 Diferenças

As diferenças são outras das estruturas de dados importantes. Através delas é possível representar a informação determinada na avaliação dos grafos.

Durante a comparação dos grafos existe um conjunto que vai acumulando as suas diferenças. No final, a interpretação desse conjunto permite enviar aos alunos um *feedback* detalhado dos seus erros. A representação de diferenças como estruturas permite também comparar as diferenças esperadas e obtida, o que é importante na validação do avaliador, como se explica no Capítulo 4.

As diferenças dividem-se em três grandes grupos: inserção, remoção e modificação.

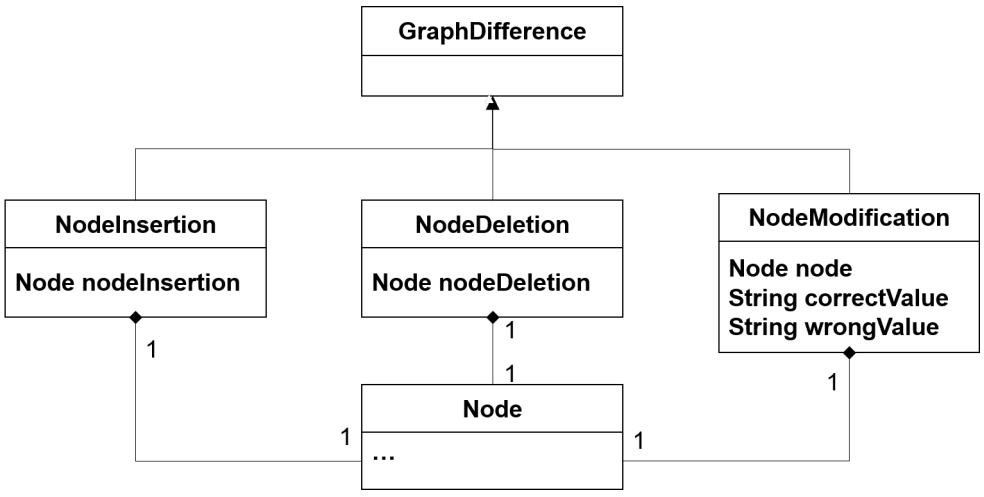

Figura 3.4: Representação esquemática das diferenças

Na Figura 3.4 surgem ilustradas as diferenças relativas aos nós, no entanto, existem estruturas semelhantes relativas aos arcos e propriedades. Seguidamente são descritos os três tipos de diferenças. De notar mais uma vez que, apesar de os exemplos retratarem apenas diferenças ao nível dos nós, existem estruturas com funcionamento análogo para os arcos e propriedades.

3.2.4.1 Inserção

As diferenças de inserção são reflexo de uma tentativa em que falta algum objeto comparativamente com a solução. Na Figura 3.5 pode-se observar que a tentativa apresenta um nó em falta e, consequentemente, também um arco em falta. Por isso essas duas diferenças devem ser refletidas no *feedback*.

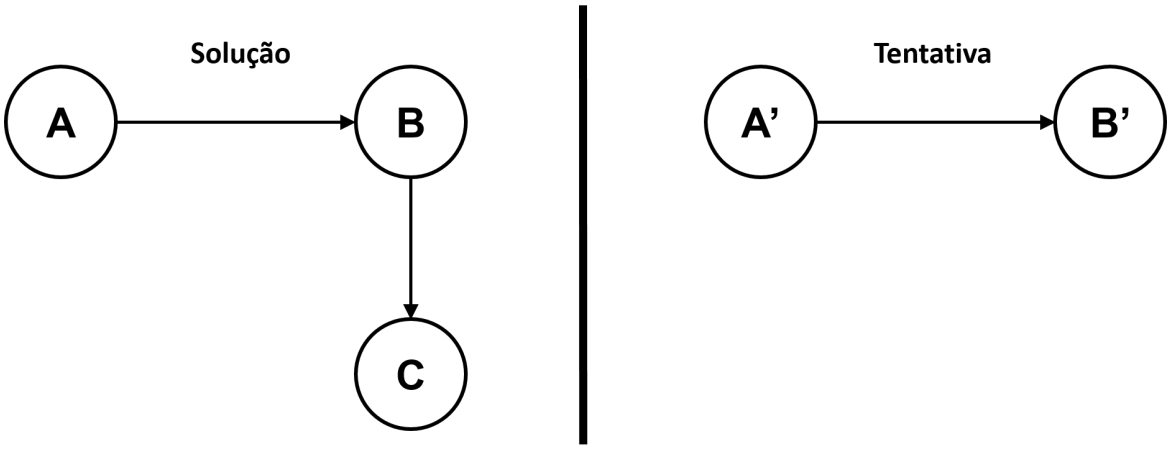

Figura 3.5: Comparação de grafos - Inserção

Nesta situação o *output* a receber pelo aluno seria algo deste tipo:

Inserir um nó do tipo ... (tipo do nó C)
Inserir um arco do tipo ... (tipo do arco B-C)

3.2.4.2 Remoção

Analogamente ao apresentado na secção anterior, caso a tentativa apresente alguns elementos que não deveriam existir, o sistema é capaz de os identificar e pedir ao aluno que os remova. Vejamos a Figura 3.6.

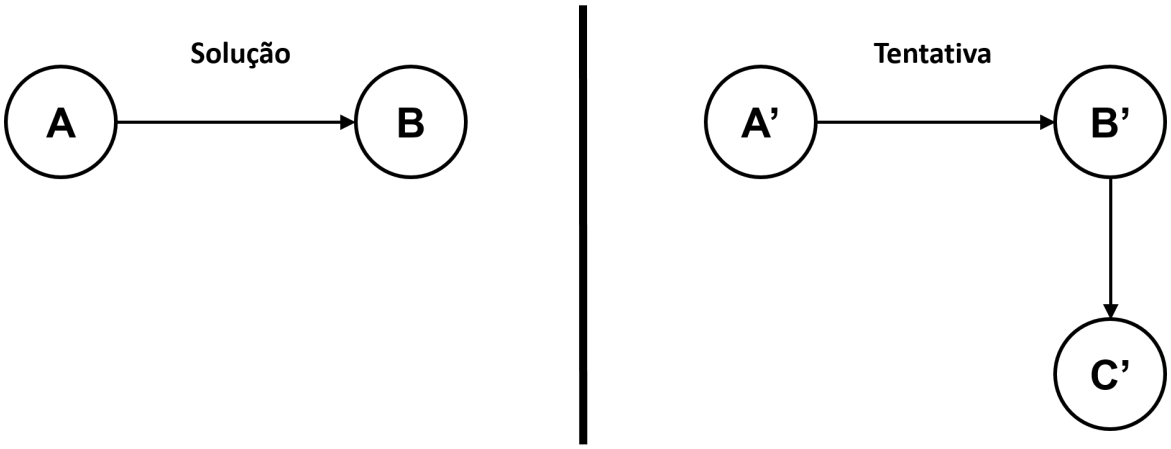

Figura 3.6: Comparação de grafos - Remoção

Como se pode observar, desta vez existe um nó e uma ligação a mais na tentativa. Pelo que nestas circunstâncias o *feedback* dado seria do tipo:

Remover o nó C'
Remover o arco B'-C'

3.2.4.3 Modificação

As diferenças de modificação indicam que existe um elemento que se encontra no local correto mas que apresenta um tipo ou valor diferente do que era esperado. Vejamos o exemplo da Figura 3.7. Assumamos que as cores dos nós representam o seu tipo.

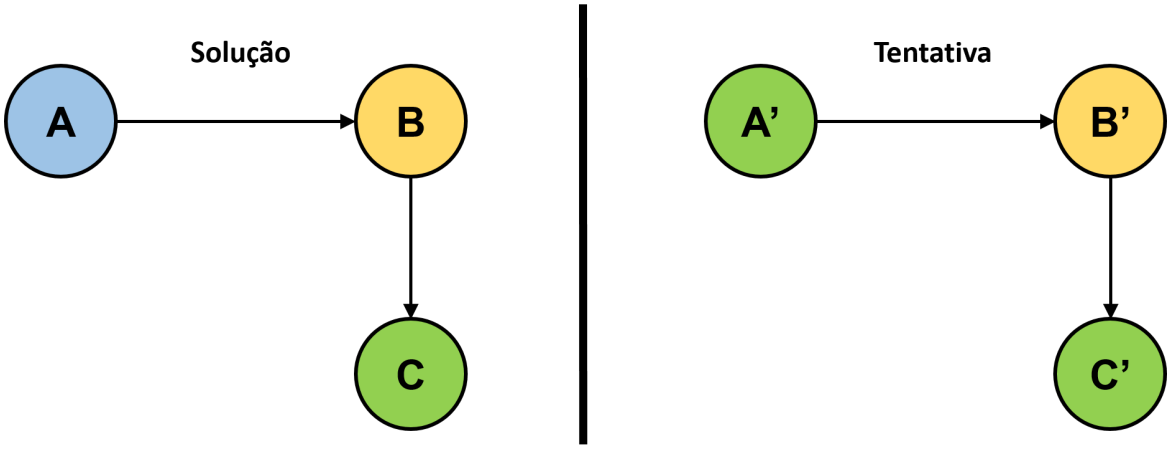

Figura 3.7: Comparação de grafos - Modificação

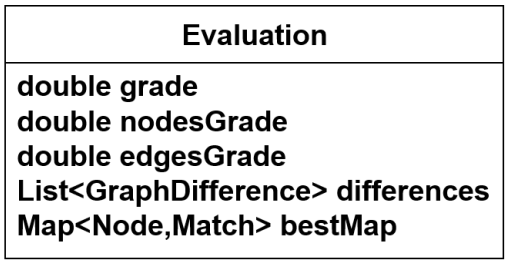

Figura 3.8: Representação da classe *Evaluation*

Agora não existe nada em falta no grafo. Ambos têm o mesmo número de nós e arcos e as conexões refletem as mesmas ligações em ambas as situações. No entanto, temos que um nó apresenta um tipo (neste caso a sua cor) diferente do esperado. Pelo que a mensagem enviada pelo sistema seria:

O nó A' está com tipo "verde" mas deveria ser "azul"

3.2.5 Avaliação

A avaliação é o resultado produzido pela execução do algoritmo. Para isso, foi criada um tipo de dados *Evaluation* que agrupa todas as componentes importantes para dar *feedback*. As duas informações mais relevantes são a melhor nota calculada e a lista de diferenças respetiva. A nota permite, ao aluno que submeteu o exercício, perceber o quão perto ou distante se encontra da solução, em termos percentuais. O conjunto de diferenças permite ao sistema produzir uma série de dicas que ajudará o alunos a melhorar a sua tentativa.

Além destas informações são guardados na avaliação os valores calculados para as notas dos nós e para as notas dos arcos, bem como o mapeamento que gerou estes resultados. Estas informações não são destinadas ao aluno, mas são essenciais para verificar se o resultado obtido foi aquele esperado. Estas verificações são feitas na validação, como é descrito no próximo capítulo.

3.3 Avaliador

O avaliador é o centro de todo este trabalho. Aqui é efetuado todo o processo de comparação e mapeamento de nós, busca de diferenças e cálculo da nota. A principal tarefa realizada antes da execução do avaliador é a conversão dos ficheiros contendo os

diagramas em grafos estendidos. A comparação pode ser dividida em fases, conforme representado pelas pistas (*swimlanes*) do diagrama UML de atividade representado na Figura 3.9.

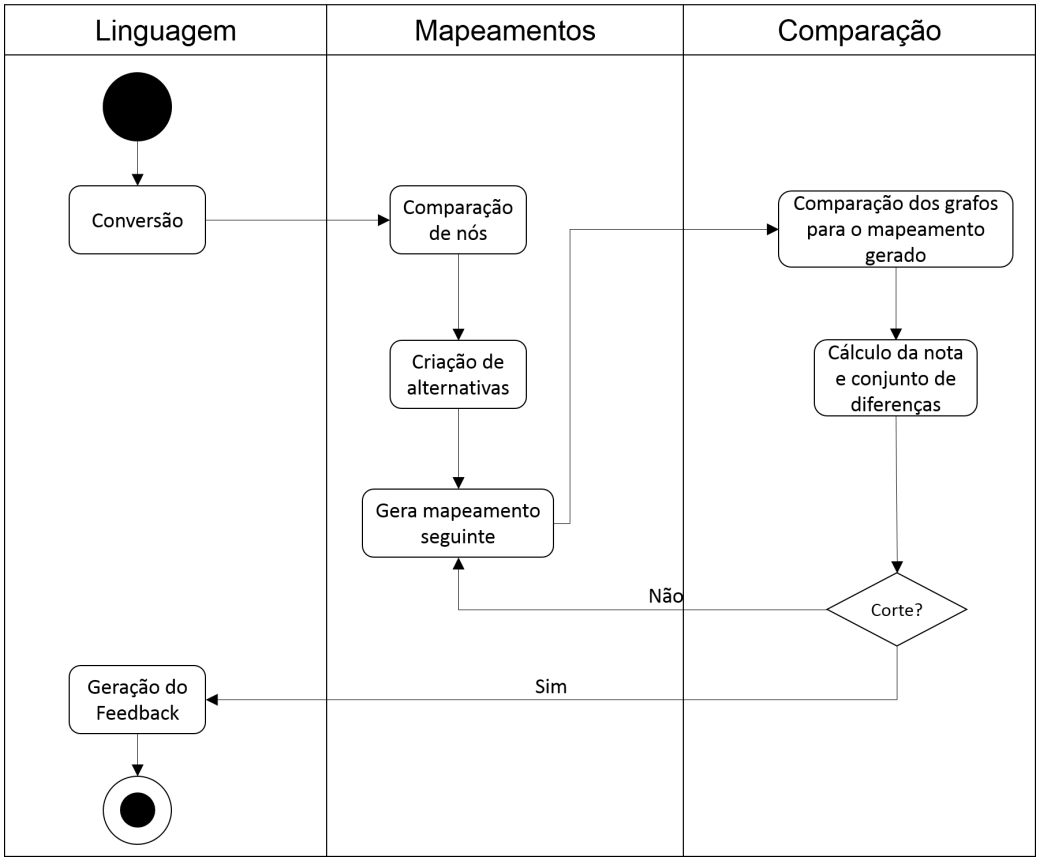

Figura 3.9: Diagrama de atividade esquematizando o funcionamento do avaliador

A primeira fase é relativa à transformação entre diagramas e grafos. Nesta fase são convertidos os diagramas em grafos estendidos. No final são convertidas as listas de diferenças em mensagens para os alunos.

A segunda fase é dedicada à geração dos mapeamentos entre os nós do grafo solução e os nós do grafo tentativa. Para cada um dos mapeamentos encontrados, será calculada uma nota e criado um conjunto de diferenças associados ao respetivo mapeamento. Este processo repete-se até que a melhor nota encontrada não possa ser melhorada. Nesse instante é retornado um objeto do tipo *Evaluation* que contém todas as informações relevantes sobre o melhor resultado encontrado. Cada um destes processos encontra-se descrito, detalhadamente, nas subsecções seguintes.

3.3.1 Gerador de mapeamentos

Como já foi referido anteriormente, pretende-se encontrar o conjunto de diferenças entre dois grafos sem gerar todas as permutações de nós possíveis. Se tivermos dois grafos com n nós, então existem $n!$ possíveis mapeamentos. Se isto não é um problema para grafos pequenos (com menos de 10 nós), à medida que n aumenta, fica incomportável a aplicação deste método.

Nós	Mapeamentos Possíveis
2	$2! = 2$
3	$3! = 6$
4	$4! = 24$
...	...
10	$10! = 3628800$
...	...
15	$15! = 1307674368000$
...	...
30	$30! = 2,6 \times 10^{32}$

Tabela 3.1: Variação do número de mapeamentos possíveis em função do número de nós

A Tabela 3.1 ilustra a rapidez do crescimento fatorial. Para comparar dois grafos com trinta nós, sem a utilizar heurísticas, exige $2,6 \times 10^{32}$ comparações entre os pares de grafos a comparar, por oposição a $3,6 \times 10^{6}$ para 10 nós.

A solução passa por desenvolver uma heurística que permita evitar todos esses testes. A abordagem seguida é a de gerar todos os possíveis mapeamentos de nós, de forma ordenada (do mais provável para o menos provável), para que seja possível utilizar um critério de corte.

3.3.2 Comparação de Nós

Antes disso necessitamos da definição de uma fórmula de cálculo que permita obter um valor numérico para comparar nós.

A opção tomada foi atribuir a cada nó e arco um valor que é calculado a partir das suas características: tipo e propriedades (e grau, no caso do nós). A fórmula de cálculo

baseia-se nos seguintes valores, em que os pesos dependem da linguagem dos diagramas considerada:

T - peso do tipo

P - peso da propriedade

p - número de propriedades

G - peso do grau

g_e - grau de entrada

g_s - grau de saída

O valor associado a cada arco calcula-se a partir da seguinte fórmula:

$$valor_{arco} = T + P \times p$$

Por sua vez, o valor de cada nó é dado por:

$$valor_{no} = T + P \times p + G \times (g_e + g_s)$$

Apesar de a explicação acima incidir sobre ambos os géneros de elementos do grafo (nós e arcos), neste fase do processo apenas necessitamos dos nós. Para ilustrar este cálculo vejamos como exemplo a Figura 3.10 que representa um nó. O nó utilizado não reflete nenhum elemento de uma linguagem em particular.

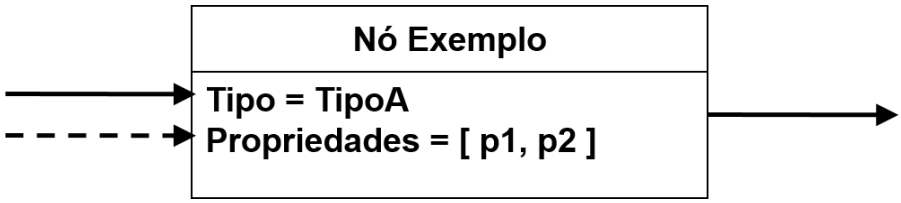

Figura 3.10: Representação de um exemplar de nó

Então vejamos qual seria o valor deste nó. Sendo que T, P e G são constantes, os únicos elementos que temos de ter em atenção são as propriedades e os graus. E nesse aspeto, temos:

$p = 2$

$g_e = 2$

$g_s = 1$

Aplicando a fórmula apresentada em cima, temos:

$$valor_{no} = T + 2P + (2 + 1)G$$

De forma a completar o exemplo, se arbitrarmos que as constantes estão definidas com os seguintes valores: $T = 10$, $P = 2$ e $G = 1$ temos:

$$valor_{no} = 10 + 2 \times 2 + 3 \times 1 = 17$$

O próximo passo é perceber como é que estes valores nos ajudam na comparação dos nós. Mais uma vez, a fórmula a utilizar é em tudo semelhante à que acabamos de ver. Para ser mais simples perceber como é calculada a comparação entre dois nós, dividimos esse valor em três campos: os mesmos que são utilizados na definição de valor de um nó (tipo, propriedades e grau). Então, sendo n um nó pertencente à solução e n' um nó pertencente à tentativa, calculamos $comparar(n, n')$ da seguinte forma:

$$comparar\_tipo(n, n') + comparar\_propriedades(n, n') + comparar\_grau(n, n')$$

A comparação dos tipos de dois nós é direta e implica apenas a atribuição de um valor: se os tipos são iguais retorna T, caso contrário retorna 0.

O cálculo mais complexo é relativo à comparação de propriedades. Para cada propriedade da solução tenta-se encontrar uma propriedade equivalente na tentativa. Caso se encontre, essa propriedade da tentativa não pode voltar a ser utilizada. No entanto, podem existir situações bem distintas.

O primeiro caso é o de n e n' terem o mesmo número de propriedades. Se todas essas propriedades de n conseguem uma correspondência nas de n', então a nota relativa às propriedades será máxima ($P \times p$) Dentro deste caso, temos ainda a situação em que nem todas as propriedades de n encontram um par em n'. Nesse caso o valor a retornar teria em conta, não todas as propriedades, mas apenas o número de propriedades em comum (pc) o que seria inferior ($pc < p \implies P \times pc < P \times p$).

Outra hipótese é n ter mais propriedades do que n', então queremos que o valor a retornar seja inferior ao máximo. Por isso relacionamos esse valor com o número de propriedades em comum e garantimos, mais uma vez, a condição que desejamos ($pc < p \implies P \times pc < P \times p$)).

Por fim, se n tem menos propriedades do que n', mais uma vez queremos que o valor a retornar seja inferior ao máximo. Porém desta vez, não basta analisar as propriedades em comum porque nesta situação temos que $pc \leq p$. Por consequência temos que $P * pc \leq P * p$. Então, no limite teríamos uma situação em que n' apresentaria todas as propriedades comuns a n e ainda mais algumas, e seria retornado o valor máximo. A solução neste caso passa por aplicar uma penalização associada ao número de propriedades que estão a mais na tentativa. Representemos estas penalizações por pm. Assim asseguramos que o valor retornado não será máximo ($P \times (pc - pm)$). Desta forma, em certos casos, esta nota pode assumir valores negativos; aconteceria no caso de o número de propriedades extra exceder o número de propriedades em comum, o que, à partida, isso apenas acontecerá em nós onde a probabilidade de serem o mapeamento um do outro é muito baixa.

Vejamos agora o cálculo da nota relativa à comparação de graus. Como já referido anteriormente, cada nó contém a informação relativa ao grau de entrada e de saída, de forma discriminada por tipos de ligação. Quer isto dizer, que existe uma atribuição do tipo chave-valor, onde a chave é o tipo da ligação e o valor representa o número de ligações desse tipo que chega ou que parte desse nó, consoante o conjunto de pares que estejamos a analisar. O valor final a retornar será a soma da comparação dos graus de entrada e de saída, que são feitos do mesmo modo.

Resumidamente, faz-se algo semelhante ao que se fez em cima com as propriedades. Conta-se o número de ligações com tipos comuns e, no caso de existirem ligações a mais do lado da tentativa, aplica-se uma penalização. Por simplicidade, tomemos como exemplo a avaliação do grau de entrada. Para cada tipo de ligação que chega ao nó solução é necessário saber quais são as quantidades quer na solução (ls) quer na tentativa (lt), de forma a se calcular o número de ligações em comum (lc) desta forma:

$$lc = ls - |ls - lt|$$

Desta forma, se $ls = lt \implies |ls - lt| = 0 \implies lc = ls = lt$ o que significa que, para o tipo de ligação em questão, existe a mesma quantidade de ambos os lados. Se isso não acontecer, a penalização $|ls - lt|$ é aplicável quer na situação em que $ls > lt$ quer

na situação em que $ls < lt$. Porém isto não é suficiente. Depois disto é necessário percorrer os tipos de ligações da tentativa e verificar se existe algum tipo que não exista na solução. Caso isso acontece, a quantidade dessa ligação é utilizada como penalidade.

Mais uma vez, devido às penalizações, esta parcela também pode assumir valores negativos. Como a comparação entre dois nós é a soma destes três componentes independentes, poderia assumir valores negativos. No entanto, optou-se por atribuir nessas situações o valor de comparação 0. Assim temos:

$$0 \leq comparar(n, n') \leq valor(n).$$

Esta fórmula é utilizada para calcular também a comparação dos arcos. Obviamente que com a diferença da nota relativa aos graus não existir nos arcos.

De forma a compreender melhor estes conceitos vejamos alguns exemplos. O exemplo mais simples é aquele em que comparamos dois nós iguais e que obteríamos a nota máxima. No entanto, para além de ser trivial, não seria possível perceber a fórmula de cálculo. Por isso, os três exemplos apresentados a seguir, apresentam todos alguma característica que permite compreender a aplicação da fórmula. Para isso, consideremos sempre n como Nó Exemplo1 e n' como Nó Exemplo2. Para além disso, utilizemos outra vez, os pesos anteriores: $T = 10$, $P = 2$ e $G = 1$.

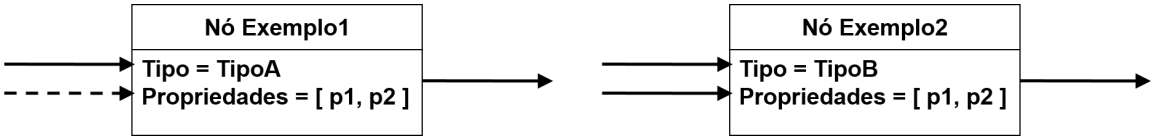

Figura 3.11: Primeiro exemplo da comparação de nós

Apliquemos então a fórmula ao exemplo da Figura 3.11 para efetuar a o seguinte cálculo: $comparar(n, n')$. Aqui temos que $comparar\_tipo(n, n') = 0$ porque os seus tipos são diferentes.

Relativamente às propriedades, temos que o número de propriedades é 2 dos dois lados e que ambas as propriedades têm uma correspondência do outro lado, então: $comparar\_propriedades(n, n') = P \times 2 = 2 \times 2 = 4$.

Por último, os graus. No nó n temos como entrada uma ligação do tipo "Seta cheia fechada" e uma ligação do tipo "Seta tracejada fechada". Na tentativa tem duas

ligações do tipo "Seta cheia fechada". Aqui vamos ver o número de ligações comuns por tipo:

1. "Seta cheia fechada": $1 - |1 - 2| = 0$
2. "Seta tracejada fechada": $1 - |1 - 0| = 0$

No grau de entrada, a nota é a soma dessas duas avaliações, neste caso 0. No grau de saída existe apenas uma ligação em cada nó e é do mesmo tipo pelo que a nota é: $1 - |1 - 1| = 1$. Somando a nota do grau de entrada com o de saída temos: $comparar\_grau(n, n') = G \times (0 + 1) = 1 \times 1 = 1$. Então a nota de comparação destes dois nós seria: $comparar(n, n') = 0 + 4 + 1 = 5$.

Vejamos agora o segundo exemplo.

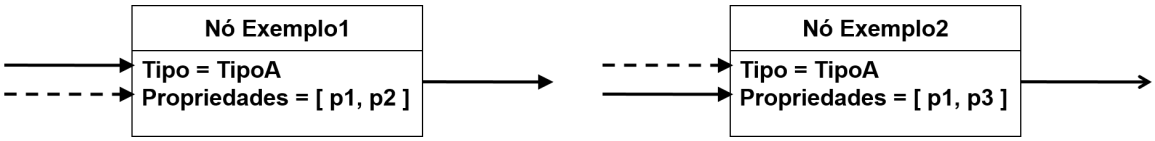

Figura 3.12: Segundo exemplo da comparação de nós

Desta vez, como podemos observar os tipos de n e n' são iguais, pelo que:

$$comparar\_tipo(n, n') = T = 10$$

Mais uma vez, temos que os conjuntos de propriedades tem o mesmo tamanho, porém só têm uma propriedade em comum, o que faz com que:

$$comparar\_propriedades(n, n') = P \times pc = 2 \times 1 = 2$$

Por fim temos a comparação de graus. Facilmente observamos que temos exatamente o mesmo tipo de ligações de lado solução e da tentativa, pelo que pela aplicação da fórmula, iremos obter que o número de ligações em comum será igual ao número de ligações da solução (ou da tentativa, visto que são iguais). Isto acontece quer para o grau de entrada, quer para o de saída. Então:

$$comparar\_grau(n, n') = G \times (2 + 1) = 1 \times 3 = 3$$

Somando tudo isto, temos:

$$comparar(n, n') = 10 + 2 + 3 = 15$$

Vejamos um último exemplo.

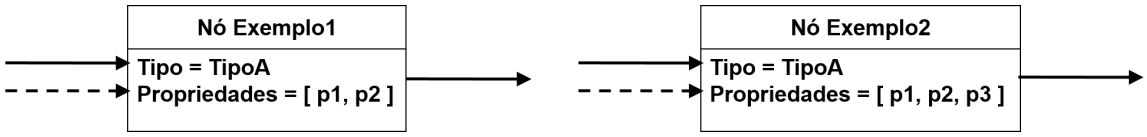

Figura 3.13: Terceiro exemplo da comparação de nós

Na Fig 3.13 vemos algumas coisas em comum com o exemplo anterior: os tipos e os graus. Pelo que teremos exatamente o mesmo valor de $comparar\_tipo(n, n')$ e $comparar\_grau(n, n')$ que obtivemos no caso da Fig 3.12, neste caso:

$$comparar\_tipo(n, n') = 10$$

$$comparar\_grau(n, n') = 3$$

A única diferença encontra-se no conjunto de propriedades. Aplicando a fórmula, teríamos de descobrir quantas propriedades existem em comum. O que, por observação, vemos que são 2. Porém, 2 é o valor máximo de propriedades em comum que podemos obter nesta situação, visto que a solução apresenta apenas duas propriedades. Neste caso, como a tentativa apresenta mais propriedades do que deveria, têm de ser aplicado uma penalização, que é igual ao número de propriedades que tem a mais (pm), neste caso: 1. Então ficaríamos com:

$$comparar\_propriedades(n, n') = P \times (pc - pm) = 2 \times (2 - 1) = 2$$

Somando tudo obtemos:

$$comparar(n, n') = 10 + 2 + 3 = 15$$

Estes dois últimos exemplos mostram que ter uma propriedade errada, a mais ou em falta terá exatamente o mesmo impacto na comparação. O mesmo se aplica aos graus.

3.3.3 Criação de Alternativas

Tudo isto é importante numa fase inicial porque o primeiro passo a executar, de forma a ser possível encontrar os mapeamentos de forma ordenada, é comparar todos os nós

do grafo solução e do grafo tentativa. Ou seja, criam-se todos os pares de nós possíveis, se os grafos têm n então, teremos n^2 pares. Esses pares, para além de agruparem o nó solução com o nó tentativa referentes, guardam também a informação relativa ao valor da sua comparação e um conjunto de diferenças entre eles (por exemplo: "Tipo Diferente" ou "'Propriedade em falta", etc.). O tipo de cada um desses pares é denominado *Match*.

Após a computação de todas as comparações, temos uma matriz $n \times n$ com os dados em questão. Ficamos com a informação como é apresentada na Tabela 3.2. O passo seguinte é, para cada nó solução, escolher o nó que mapeia com valor mais alto. Na Tabela 3.2 esses mapeamentos estão assinalados a negrito.

Tentativa / Solução	A'	B'	C'	D'
A	**12**	8	11	3
B	8	**15**	7	6
C	10	7	**11**	0
D	3	6	0	**17**

Tabela 3.2: Exemplo de uma tabela de comparações dos nós solução com os nós tentativa

Com esses *Matches* com valor mais elevado, construi-se um mapeamento que se considera "o melhor". Com "o melhor" quer-se dizer que não existe nenhum outro mapeamento cuja soma das comparações de todos os seus pares seja superior a esta. No entanto, não se pode ainda prever a influência que os arcos terão no cálculo da nota nesta fase.

Então temos o seguinte, neste caso:

Match(A,A',12)	*Match*(B,B',15)	*Match*(C,C',11)	*Match*(D,D',17)

Tabela 3.3: "Melhor" mapeamento com base no exemplo da Tabela 3.2

O passo seguinte passa por calcular, para todos os outros pares alternativos, a diferença entre o valor que esse nó solução possui no seu melhor *Match* e o seu valor de comparação desse par. Por fim, devem ser colocadas numa lista por ordem crescente dessa diferença.

Para ser mais percetível, peguemos no exemplo do nó A. O melhor mapeamento encontrado foi com A'. No entanto, A pode mapear também com B', C' ou D'. Por isso, vamos calcular a diferença entre o melhor mapeamento da A e essas comparações. Vejamos:

$comparar(A, A') = 12$
$melhor(A) - comparar(A, A') = 12 - 12 = 0$

$comparar(A, B') = 8$
$melhor(A) - comparar(A, A') = 12 - 8 = 4$

$comparar(A, C') = 10$
$melhor(A) - comparar(A, A') = 12 - 10 = 2$

$comparar(A, D') = 3$
$melhor(A) - comparar(A, A') = 12 - 3 = 9$

Neste exemplo, teríamos de efetuar os mesmos cálculos para os nós B, C e D, obtendo os valores que se encontram entre parênteses na Tabela 3.4.

Tentativa / Solução	A'	B'	C'	D'
A	**12** (0)	8 (4)	11 (1)	3 (9)
B	8 (7)	**15** (0)	7 (8)	6 (9)
C	10 (1)	7 (3)	**11** (0)	0 (10)
D	3 (15)	6 (11)	0 (17)	**17** (0)

Tabela 3.4: Exemplo de uma tabela de comparações dos nós solução com os nós tentativa e respetiva diferença para o melhor mapeamento

Tal como sugerido em cima, a tarefa seguinte passa por ordenar todas as alternativas ao "melhor" mapeamento de forma crescente relativamente à diferença. Isto faz com que as primeiras alternativas sejam as que apresentam menor diferenças. Estas alternativas são utilizadas para gerar os mapeamentos seguintes. Vejamos como seria a lista ordenada de alternativas para o exemplo que estamos a considerar.

Match(C,A',10)	*Match*(A,C',11)	*Match*(C,B',7)	...	*Match*(D,C',0)
Diferença: 1	Diferença: 1	Diferença: 3	...	Diferença: 17

Tabela 3.5: Lista ordenada de alternativas

Esta lista tem como objetivo ser utilizada para efetuar trocas do mapeamento a avaliar. Por isso, como o primeiro mapeamento a avaliar é o "melhor", todos esses *Matches* são excluídos desta lista, pois caso contrário, como esses apresentariam diferenças de zero, seriam os primeiros da lista e estar-se-ia a efetuar trocas desnecessárias. Como se efetuam as trocas é explicado na subsecção seguinte.

3.3.4 Escolha do Mapeamento

É necessário gerar um mapeamento de nós solução em nós tentativa sempre que este for pedido. Queremos que a soma das comparações dos nós nesse mapeamento seja sempre decrescente. Para isso temos que ir efetuando as trocas necessárias pela ordem da lista, o que pode ser ilustrado com o exemplo da subsecção anterior (Tabela 3.5.

Começamos com o primeiro mapeamento que é o "melhor" possível. Como já vimos, na Tabela 3.3 se somarmos os valores de comparação obtemos $12 + 15 + 11 + 17 = 55$. Então o mapeamento que queremos de seguida é aquele que produza a menor diferença possível a essa soma. Vamos verificando se é possível criar mapeamentos que gerem diferenças sucessivas de $0, 1, 2, ..., s$. Sendo que s é a soma das comparações do "melhor" mapeamento, neste exemplo 55.

Olhando para as alternativas vemos que não é possível efetuar trocas que produzam diferença zero. No entanto, efetuando uma troca com a primeira alternativa obtemos diferença 1, isto é, passaríamos de 55 para 54. Para além disso é necessário verificar se essa troca torna o mapeamento inválido ou não. Um mapeamento inválido é um mapeamento que associa mais do que um nó da solução ao mesmo nó tentativa, ou o contrário. Observemos o que obteríamos na primeira situação:

Match(A,A',12)	*Match*(B,B',15)	***Match*(C,A',10)**	*Match*(D,D',17)

Tabela 3.6: Segundo mapeamento gerado

Como a primeira alternativa é um *match* de C em A', o que se tem de fazer é trocar o no "melhor" mapeamento o *match* de C, que no caso era em C'. Essa troca está assinalada a negrito na Tabela 3.6. Porém, como se pode ver, neste mapeamento existem dois nós solução (A e C) que mapeiam no mesmo nó tentativa (A'). Então este mapeamento é inválido e tem de ser procurar o próximo.

Como ainda existe outra alternativa que diminui a soma em 1, é essa a ser testada.

***Match*(A,C',11)**	*Match*(B,B',15)	*Match*(C,C',11)	*Match*(D,D',17)

Tabela 3.7: Terceiro mapeamento gerado

Desta vez, a troca foi efetuada no mapeamento relativo a A, que deixa ter correspondência com A' e passa a ter com C' (ver Tabela 3.7). Porém, esta troca torna o mapeamento novamente inválido pelas mesmas razões de cima: existem dois nós solução (A e C) que mapeiam no mesmo nó tentativa (C').

Como não existem mais alternativas que causem diferença de 1, o passo seguinte é procurar mapeamentos com diferença de 2. Observando a lista de alternativas, à primeira vista diríamos que não existe essa possibilidade porque a alternativa seguinte causa diferença de 3. No entanto, se utilizarmos as duas primeiras em simultâneo estaremos a causar a diferença que desejamos: $1 + 1 = 2$. O que obtemos neste caso é o seguinte:

***Match*(A,C',11)**	*Match*(B,B',15)	***Match*(C,A',10)**	*Match*(D,D',17)

Tabela 3.8: Quarto mapeamento gerado

Aqui trocamos o primeiro e terceiro pares, e podemos verificar que não existem repetições de nós no mapeamento, o que permite considerar este mapeamento como válido e pedir para que seja avaliado.

Este processo de geração de mapeamentos é feito sempre que o avaliador pede um mapeamento. Ou seja, não são todos gerados inicialmente nem têm de vir a ser todos gerados, visto que a certa altura o critério de corte do avaliador será ativado e não serão pedidos mais mapeamentos.

3.3.5 Cálculo da nota

Vejamos agora como é calculada a nota de comparação de dois grafos. Por definição, optou-se por estabelecer que $0 \leq nota \leq 100$.

Figura 3.14: Representação esquemática da nota de comparação entre dois grafos

Observemos a Figura 3.14. Aqui temos as duas formas em que o resultado da comparação é guardado. A primeira forma, toda a cinza, representa a nota máxima global. De forma colorida, é apresentada também a nota máxima, mas dividida em dois parâmetros: nota da comparação dos nós (azul) e nota da comparação dos arcos (vermelho).

Como já foi afirmado em secções anteriores, para efeitos de corte é importante que o peso atribuído à nota dos nós seja superior à dos arcos. O porquê está explicado detalhadamente mais abaixo. No exemplo desta figura temos que 60% da nota global é relativa aos nós e os restantes 40% relativa aos arcos.

Sejam P_n e P_a o peso que nós e os arcos têm no cálculo da nota, respetivamente. Vejamos então como se calculam os dois parâmetros.

Comecemos pela nota dos nós. Este cálculo é efetuado de forma muito simples. Para cada mapeamento dado, a nota dos nós é igual à soma dos mapeamentos a dividir pelo soma dos valores de cada nó. A esse valor é multiplicado o peso associado:

$$nota\_dos\_nos = P_n \times \frac{\sum\limits_{n \in N} k}{\sum\limits_{n \in N} valor(n)},$$

onde N representa o conjunto de nós do grafo solução e k representa o valor associado ao mapeamento em questão para o nó n. Fica mais fácil a compreensão através de um exemplo. Vejamos o seguinte:

Nó	Valor	Mapeamento
A	15	*Match*(A,A',13)
B	9	*Match*(B,B',9)
C	12	*Match*(C,C',11)

Tabela 3.9: Tabela com dados dos nós solução e seus mapeamentos

Apliquemos então a fórmula apresentada acima com os dados do exemplo da Tabela 3.9:

No numerador ficamos com $13 + 9 + 11 = 33$

No denominador ficamos com $15 + 9 + 12 = 36$

Então a nota dos nós é:

$$nota\_dos_nos = P_n \times \frac{33}{36} = 0,91(6)P_n$$

Usando o $P_n = 60$ (valor utilizado no exemplo da Figura 3.14) temos:

$$nota\_dos\_nos = 0,91(6) \times 60 = 55$$

Relativamente aos arcos, apesar da fórmula de cálculo ser análoga, os valores não se encontram pré-computados. O que é necessário fazer para estas situações é, em primeiro lugar, recorrendo ao mapeamento em questão, verificar quais as ligações que mapeiam entre si. Ou seja, os arcos que ligam os nós n_1 e n_2 na solução têm de ser comparados com os arcos que ligam $map(n_1)$ e $map(n_2)$ na tentativa. Sendo que $map(n)$ é a função que dado um nó solução n devolve um nó tentativa no qual n está mapeado. Essa comparação é feita segundo a fórmula apresentada na Secção 3.3.1. Após isso é aplicada a seguinte fórmula:

$$nota\_dos\_arcos = P_a \times \frac{\sum\limits_{e \in E} k}{\sum\limits_{e \in E} valor(e)},$$

onde E representa o conjunto de arcos do grafo solução e k representa o valor associado à comparação de cada arco com o respetivo da tentativa.

À medida que é feita esta comparação, são adicionadas as diferenças encontradas ao conjunto de diferenças relativas ao mapeamento.

Em suma temos que:

$$0 \leq nota\_dos\_nos \leq P_n$$

$$0 \leq nota\_dos\_arcos \leq P_a$$

$$0 \leq nota \leq P_n + P_a$$

3.3.6 Critério de Corte

Esta fase do avaliador é extremamente importante. Neste passo é verificado se é possível melhorar o resultado atual. Caso contrário pode-se terminar a execução com segurança, evitando gerar e testar a maioria dos mapeamentos.

Vejamos então como pode ser aplicado o critério de corte. Relembrando que o aspeto fundamental é que os mapeamentos são dados de forma ordenada, de modo que a sua soma é cada vez menor (ou igual), o que leva a que a nota dos nós seja ela, também, menor (ou igual) a cada iteração. Isto acontece porque na fórmula $P_n \times \frac{\sum_{n \in N} k}{\sum_{n \in N} valor(n)}$, o único valor que não é uma constante é o numerador da fração, visto que é o valor que depende do mapeamento. Como esse valor é decrescente a cada iteração, então também o seu resultado final será menor.

O critério de corte é testado a cada iteração. De cada vez que é verificada a condição é necessário saber qual a melhor nota (M) encontrada até esse momento e qual a nota dos nós para o mapeamento atual. Posto isto, é verificado se M é maior do que a nota global calculada a partir da nota dos nós e do máximo dos arcos. Se o teste for positivo, então não é necessário testar novos mapeamentos visto que será impossível melhorar a nota que já se encontrou.

Por exemplo, se para um mapeamento temos que $nota\_dos\_nos = 50$, então sabemos, antes de analisar os arcos, que no máximo, para este mapeamento, a nota global será igual a $50 + P_a$. Se tivermos $M = 80$ e $P_a = 40$, vemos que a nota global que no limite pode ser atingida é ainda superior à melhor encontrada até ao momento ($80 < 50+40$). Então os arcos têm de ser analisados e não se pode efetuar corte.

Porém, se nas mesmas condições tivermos um mapeamento cuja $nota\_dos\_nos = 35$ então o corte será ativado, visto que no limite, a nota global será $35 + 40 = 75$ o que é inferior à melhor atual (80). Ou seja, a partir deste momento, todos os mapeamentos seguintes não conseguirão melhorar o resultado que é tido como "melhor atual". O gráfico da Figura 3.15 ilustra esse comportamento.

No gráfico em questão, cada barra corresponde a uma nota de um mapeamento específico. Cada uma das cores têm significados diferentes:

- A azul (em cheio) está representado a componente nota_dos_nós para cada mapeamento.
- A azul (e branco) está representada a percentagem que a nota_dos_nós perdeu

em relação ao seu máximo nesse mapeamento.

- A vermelho (em cheio) está representado a componente nota_dos_arcos para cada mapeamento.
- A vermelho (e branco) está representada a percentagem que a nota_dos_arcos perdeu em relação ao seu máximo nesse mapeamento

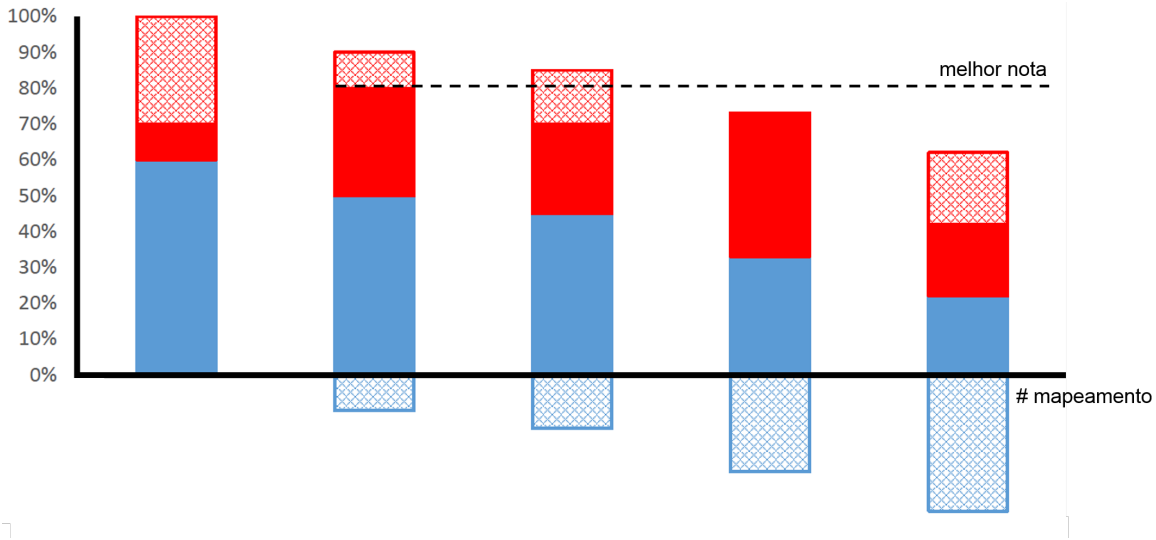

Figura 3.15: Gráfico da evolução da nota ao longo dos mapeamentos

A soma das quatro notas corresponde à nota máxima (100).

Tal como o esperado, a componente da nota dos nós vai decrescendo ao longo os mapeamentos. Podemos ver que a nota dos arcos não está diretamente relacionada com a nota dos nós. Isto é, apesar da nota dos nós ser decrescente, a nota dos arcos pode crescer.

Vejamos como é feito este processo passo a passo, sabendo que inicialmente a melhor nota até ao momento é 0. Para o primeiro mapeamento, como $M = 0$, qualquer nota será superior a esse valor. Então não se aplica corte e o valor encontrado no primeiro mapeamento é colocado em M, neste caso, $M = 70$.

Para o segundo mapeamento, vemos que a soma da nota dos nós com o máximo dos arcos é 90. Ou seja, como é superior a M, mais uma vez não existe corte. Calcula-se então os mapeamentos dos arcos e encontra-se a nota global de 80. Como é superior a M, então atualiza-se para $M = 80$.

Na iteração seguinte, no limite ainda se pode ultrapassar a melhor nota atual, por isso não se ativa o corte. No entanto, após a computação dos arcos, verifica-se que a nota não melhora pelo que valor de M não é alterado.

No final da quarta iteração vemos que, mesmo no melhor caso, será impossível melhorar a nota atual, visto que no máximo a nota calculada nesta situação será 75. Então pode ser aplicado o corte, com a certeza de que tudo o que viria a seguir não poderia melhorar o resultado obtido. Após isso, pode-se retornar o melhor resultado encontrado.

A importância dos pesos atribuídos à nota dos nós e dos arcos é visível neste situação. Isto porque quanto maior for o peso dos arcos, mais tarde será efetuado o corte. Vejamos que para aplicação do critério é sempre utilizado o valor máximo possível da nota dos arcos, então quanto maior for esse valor, maior será o valor calculado para o máximo possível. Com isso, será mais difícil chegar ao ponto em que esse valor é inferior à melhor nota encontrada em cada momento.

No entanto, de forma a que o avaliador não fique por um período indefinido em execução, quando não consegue certificar-se que já encontrou a melhor solução, foi definido um limite de tempo (*timeout*) de 2 segundos, após o qual a execução é terminada. Nessas situações dizemos que a avaliação não foi bem-sucedida (ou incompleta). Isto não quer dizer que a solução encontrada não foi a melhor possível, significa apenas que não podemos garantir com certeza de que o resultado obtido ao fim desse período é o melhor.

3.4 Grafos de Diferente Dimensão

Anteriormente foi apresentada a forma como o avaliador lida na comparação de grafos com o mesmo número de nós. No entanto, nem sempre será essa a situação. Caso fosse aplicado o algoritmo da mesma forma, nem sempre seria possível encontrar solução. A Figura 3.16 mostra um exemplo.

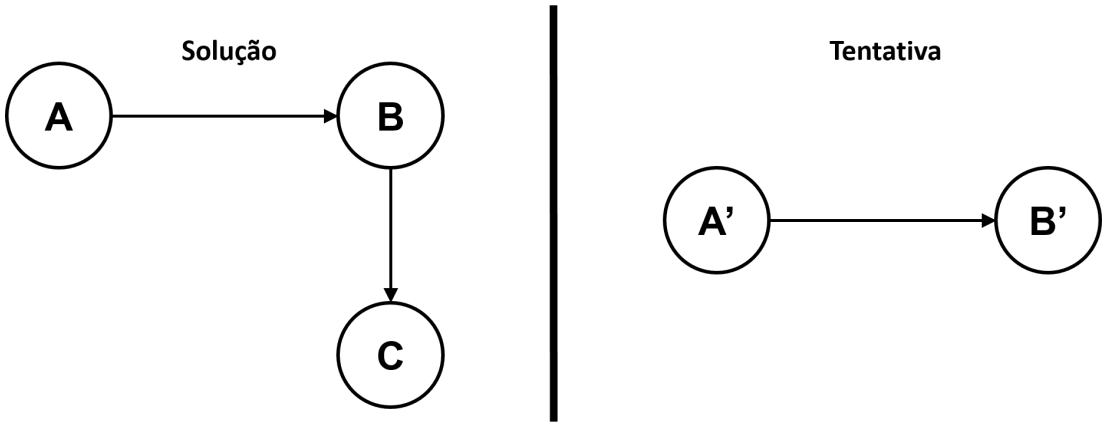

Figura 3.16: Exemplo de dois grafos com número de nós diferente

Como se pode ver na figura, a tentativa possui um nó a menos que a solução. No entanto, o passo inicial, da comparação de todos os nós solução com todos os nós tentativa é, mesmo assim, efetuado, tal como a escolha do mapeamento inicial. Vamos assumir que os valores das comparações dos nós são os apresentados na Tabela 3.10

Tentativa / Solução	A'	B'
A	**12**	8
B	8	**15**
C	**5**	3

Tabela 3.10: Exemplo de uma tabela de comparações dos nós de dois grafos com número de nós diferentes

Por observação da tabela facilmente se identifica um grande problema. Como o número de nós da solução é maior que o número de nós da tentativa, será impossível criar um mapeamento válido. Isto porque, por mais alternativas que se explorem, existirá sempre dois nós da solução que mapearão no mesmo nó tentativa.

A solução passa pela redução do grafo maior. Quer-se dizer com isto que, de forma a se utilizar o algoritmo tal com está implementado, a melhor solução é tornar os dois grafos com o mesmo número de nós.

Para isso, basta selecionar o grafo com maior número de nós e ir removendo até que os dois grafos tenham o mesmo tamanho. À medida que são removidos os nós, também são eliminadas todas as ligações que lhes estão associadas. No entanto, isso obrigaria

a testar $C_m^n = \frac{n!}{m!(n-m)!}$ pares de grafos, sendo que n é o número de nós do grafo maior e m o número de nós do grafo menor.

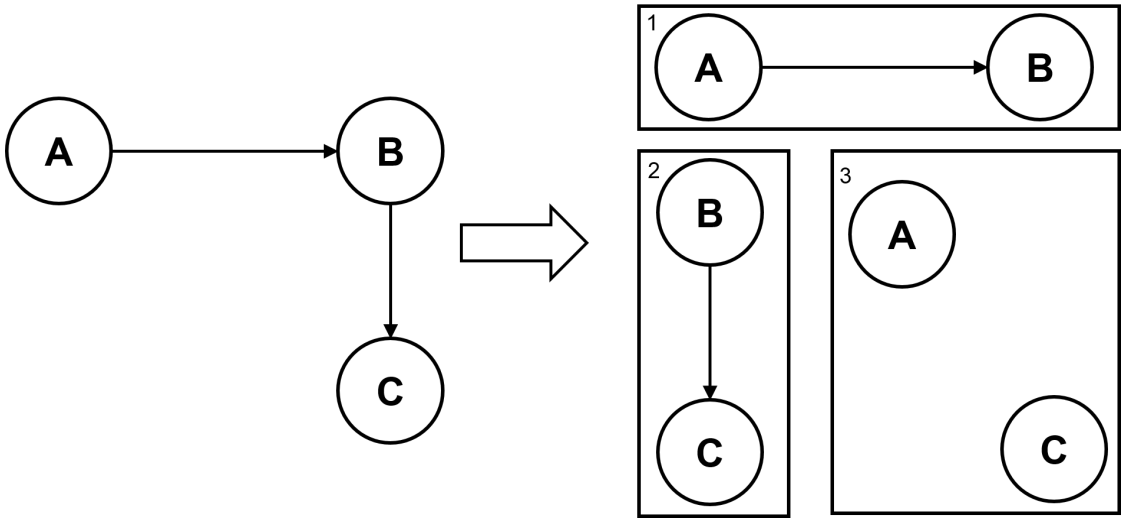

Figura 3.17: Possibilidades de redução do grafo

A Figura 3.17 ilustra as possibilidades de redução do grafo solução para o exemplo visto na Figura 3.16. Pretende-se evitar o teste de todos os grafos possíveis. Por isso, tal como feito nos mapeamentos, a escolha dos nós a remover é feita com base nos que contribuem menos para a nota. Ou seja, por análise do mapeamento inicial serão removidos em primeiro lugar os que possuem um *Match* de valor reduzido. Voltando ao exemplo que estamos a analisar temos que o mapeamento inicial é o seguinte:

Match(A,A',12)	*Match*(B,B',15)	*Match*(C,A',5)

Tabela 3.11: Mapeamento (inválido) inicial com base no exemplo da Tabela 3.10

A primeira redução a ser efetuada seria a remoção do nó C. Em seguida seria, mantendo C, remover o nó A. E por fim, a última alternativa, seria ficar com os nós A e C. Esta ordem está indicada na Figura 3.17 nos números das caixas.

No caso da redução necessária ser superior a um nó, tem de ser ir combinando, ordenadamente, os nós que geram uma contribuição menor para a classificação. Depois de reduzido um dos grafos e de ambos terem o mesmo número de nós, pode executar-se o algoritmo normalmente. O facto desta redução ser feita de forma ordenada continua a permitir a utilização do critério de corte, o que faz com que não seja necessário testar todos os pares de grafos possíveis.

3.5 Afinador de Parâmetros

O cálculo da classificação da comparação de dois grafos está dependente de um conjunto de parâmetros, como apresentado nas Secções 3.3.1 e 3.3.5. Relembrando, para calcular a nota global é necessário definir o peso atribuído à nota dos nós e a nota dos arcos. Para além disso é necessário, para cada uma dessas componentes, definir os pesos relativos ao tipo, às propriedades e, para os nós, ao grau. Porém, estes pesos não têm de ser iguais para os arcos e para os nós. Portanto, no total existem sete parâmetros que devem ser afinados, de forma a se aproximar de um avaliador humano. São eles:

- Peso dos nós
- Peso dos arcos
- Peso do tipo do nó
- Peso do tipo do arco
- Peso da propriedade do nó
- Peso da propriedade do arco
- Peso do grau do nó

Estes pesos terão de ser utilizados com diferentes valores para diferentes tipos de diagramas. Por isso, para cada tipo de grafo que queiramos avaliar, o afinador de parâmetros deve ser executado. A execução deste afinador é completamente independente do avaliador e é utilizada apenas uma vez para cada tipo de diagrama, de forma a atribuir os seus pesos. Posteriormente, esses valores ficam armazenados e podem ser utilizados a qualquer altura numa qualquer avaliação desse tipo.

O objetivo do afinador foi definir estes parâmetros de forma a que a avaliação fosse o mais semelhante possível à avaliação de um professor. Para tanto decidiu-se recorrer à utilização de um algoritmo genético, por forma a explorar um espaço vasto de soluções. A descrição desse algoritmo será muito breve, visto que a não fez parte deste projeto. Utilizamos um algoritmo existente, previamente implementado, adaptando-o às nossas necessidades.

O algoritmo genético codifica um conjunto de pesos como sendo cromossoma de um indivíduo. Foi definida uma função de aptidão (*fitness*) para esses indivíduos

comparando as avaliações resultantes desses pesos para um conjunto de diagramas, cuja nota foi previamente atribuída por um painel de peritos. A função de aptidão é a soma das diferenças de avaliação (erro) obtidas na comparação do diagrama tentativa e respetiva solução. A partir dessa função é possível selecionar os indivíduos mais aptos de cada geração, sendo a geração inicial obtida aleatoriamente. A geração seguinte é obtida combinando os cromossomas de pares de indivíduos e introduzindo algumas mutações. Essa geração é ordenada usando a função de adequação, sobrevivendo e reproduzindo-se apenas os indivíduos mais aptos.

Neste projeto decidiu-se, inicialmente, que o avaliador seria preparado para avaliar dois tipos de diagramas UML: de classes e de casos de uso. Por isso e para que o algoritmo pudesse encontrar os parâmetros mais corretos, a nota esperada para cada par de grafos a testar deveria ser o mais próximo do real possível. A solução encontrada foi a criação de um inquérito[1] que apresenta dois exercícios: um de diagrama de classes e outro de casos de uso. Para cada um deles é apresentada a solução e um conjunto de respostas erradas, onde estão destacadas as diferenças para a solução. Para cada uma dessas respostas erradas é pedida uma classificação ente 0 e 100. Desta forma, pretende-se verificar qual a importância de cada erro na classificação. Como nem qualquer pessoa é capaz de fazer este tipo de avaliação, o questionário foi enviado apenas a professores universitário que lecionam unidades curriculares onde são abordados este tipo de exercícios. O inquérito foi enviado para professores de várias partes do mundo: Portugal, Espanha, França, Itália, Reino Unido, Bélgica, Suíça, Hong Kong, Brasil, Chile e EUA.

Após a recolha das respostas foi calculada a classificação média feita pelos especialistas. Essa média foi utilizado como sendo a nota esperada para cada par de grafos. Com isso, foi possível definir os parâmetros desejados.

Contudo, os parâmetros computados não permitiram o cálculo de uma nota tão próxima do esperado como desejado. Em média, para todos os pares de grafos utilizados no questionário, obteve-se que classificação do avaliador dista cerca de 15% do desejado.

Isto poderá ter algumas explicações. A primeira delas está relacionado com o baixo número de respostas obtidas com o inquérito. Desta forma, não é possível assumir a amostra como significativa e, portanto, podemos ter definido os valores errados como

[1] Acessível em:
`https://docs.google.com/forms/d/1RoqdKcWX5CIKBlgHzwPoyYOMVhqNypsTAxLb3Olt-8U/viewform?usp=send\_form`

sendo os esperados.

Outro dos motivos que poderá explicar isso é o facto de existirem algumas discrepâncias nas respostas. Por exemplo, numa das respostas os valores oscilam entre 50% e 80%.

Existe ainda a possibilidade de cada professor atribuir pesos diferentes a objetos de diferentes tipos. A nossa abordagem contempla que todos os tipos nós terão o mesmo peso entre si, e com cada tipo de arco tipo de arco passa-se o mesmo. No entanto, por exemplo, num diagrama de casos de uso pode-se assumir que um agente tem mais importância do que um caso de uso, pelo que a falta de um agente será mais penalizado do que um caso de uso. Na nossa abordagem, como todos os tipos têm o mesmo peso, a falta de um ou de outro teria o mesmo impacto. Este poderá ser outro dos pontos que leva à diferença obtida entre o valor esperado e o calculado pelo avaliador.

3.6 Conversor de Diagramas em Grafos

Nesta secção está descrita como é feita a transformação dos diagramas em grafos estendidos. Para além disso, é apresentado o editor de diagramas escolhido e os critérios que levaram à sua escolha.

3.6.1 Editor de Diagramas

A conversão de diagramas em grafos requer como primeiro passo a escolha do editor de diagramas que permite a criação de diagramas no computador. Após a análise de algumas opções a escolha recaiu sobre o *Dia Diagram Editor* [Dia].

Editor	Livre	Vários tipos de diagramas	Online	Exige conta
Dia	Sim	Sim	Não	Não
MS Visio	Não	Sim	Não	Não
StarUML	Sim	Não	Não	Não
Papyrus	Sim	Não	Não	Não
Gliffy	Sim	Não	Sim	Sim
LucidChart	Não	Sim	Sim	Sim
Cacoo	Sim	Sim	Sim	Sim

Tabela 3.12: Tabela com parâmetros de escolha do editor de diagramas

As características que se analisaram nos editores estão resumidas na Tabela 3.12 segundo quatro parâmetros: o *software* ser de utilização livre, a capacidade de representar vários tipos de diagramas, funcionar em *desktop* ou *online* e exigir a criação de conta.

O editor deve ter a capacidade de representar vários tipos de diagramas e ser de utilização livre. Este ponto elimina o *MS Visio* pois não é de distribuição gratuita. Os outros dois programas (*StarUML* e *Papyrus*) são especializados em diagramas UML e, por isso, perdem na capacidade de representação de diagramas.

Foi considerada a hipótese de utilizar sistemas de edição de diagramas *online*. Existe um vasto leque de opções deste género, no entanto nem todas têm capacidade de representar múltiplos tipos de diagramas. De todos os editores analisados, os que apresentam as características desejadas são o *Gliffy*, o *LucidChart* e *Cacoo*. No entanto, todos eles exigem a criação de uma conta. Este facto é um entrave à utilização em contexto pedagógico e por isso é contabilizado como uma desvantagem. Para além disso, o *LucidChart* tem a desvantagem de a conta ser paga.

Por tudo isto e pelo facto de no nosso Departamento já se encontrar instalado o *Dia* em todas as máquinas, optou-se por este *software*. O conversor é um dos elementos importantes desta implementação visto que permite a transformação da informação descrita nos ficheiros que contém os diagramas em grafos estendidos.

3.6.2 Conversão de Diagramas em Grafos

O editor *Dia* permite gravar os diagramas no seu próprio formato (.dia). Este ficheiro tem a estrutura apresentada na Figura 3.18. Esta estrutura foi obtida por "*reverse engineering*" devido à falta de documentação relativa ao editor.

O elemento de topo deste tipo de documento é o *<dia:diagram>* . Este elemento tem dois filhos: *<dia:diagramdata>*, que contém um conjunto de atributos relacionados com o espaço onde está contido o diagrama (como o limite das margens) que não contribuem para a construção do grafo estendido (e por isso não estão representados no diagrama), e o *<dia:layer>* que marca o início das especificações do diagrama em si. Sempre que é reconhecida esta anotação é inicializado um grafo com os conjuntos de nós e de arcos vazios.

O elemento *<dia:layer>* pode ter vários filhos, sendo que todos eles são do tipo *<dia:object>*. Este tipo pode representar um nó ou um arco, portanto a cada entrada

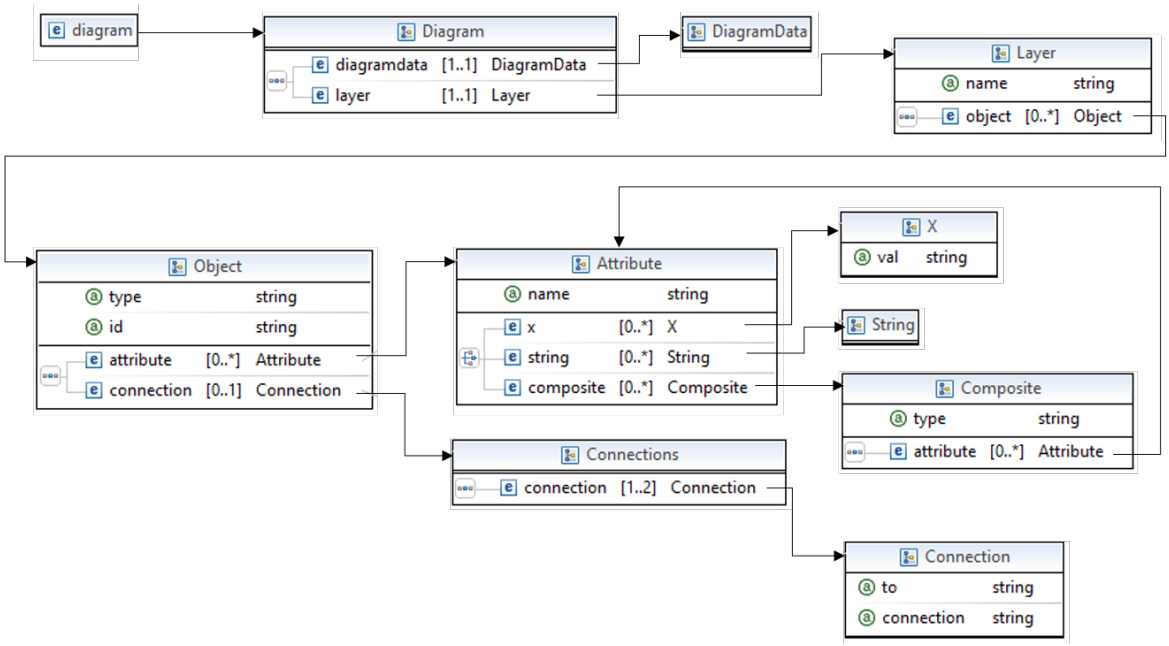

Figura 3.18: Diagrama XSD do ficheiro .dia

numa destas *tags* deve ser criado um *GObject* com o tipo e o id que surgem como atributos. Cada elemento destes pode ter vários filhos, múltiplos do tipo *<dia:attribute>* e apenas um do tipo *<dia:connections>*. Este último só aparece se o objeto for um arco. Sempre que é reconhecido um destes elementos é trocado o tipo do objeto de *GObject* para *Edge*. Este elemento apresenta no máximo dois filhos do tipo *<dia:connection>* que contém a informação relativa aos nós extremidade desta ligação.

Relativamente ao elemento *<dia:attribute>* este pode assumir vários tipos. O tipo está referido como atributo, contudo o seu valor pode assumir diversas formas. Se for um atributo simples, apenas tem um filho que contém o respetivo valor. Esse filho pode ser do tipo *<dia:X>* onde X pode ser: "*boolean*", "*point*", "*color*", etc. Nestas situações o valor encontra-se no atributo "*val*", ou num filho do tipo *<dia:string>*, sendo que neste caso o valor surge como texto.

Mais complexo é o caso de um atributo composto. Nestas situações, o filho será do tipo *<dia:composite>* que tem o seu tipo com atributo. Para além, disso, este elemento pode ter vários filhos do tipo *<dia:attribute>*.

Quando é reconhecida a anotação de fecho do objeto, este é adicionado ao grafo. Para isso é necessário verificar se o elemento tem um descendente *<dia:connections>*. Nesse caso é um arco e deve ser adicionado à lista de arcos. Caso contrário, é adicionado à lista de nós.

Quando finalmente é reconhecida a anotação de fecho do *<dia:layer>* então não existem mais elementos no grafo e pode ser dada como finalizada a sua construção.

Capítulo 4

Validação

Este capítulo descreve as duas experiências que foram realizadas de forma a validar o funcionamento do avaliador. A primeira experiência foi realizada com base em dados sintéticos e procedimentos de teste. Para a segunda experiência foram utilizados exercícios reais de linguagens de diagramas e utilizadores concretos.

4.1 Gerador de Dados Sintéticos

De forma a ser possível verificar a qualidade do algoritmo implementado, é necessário analisar o seu comportamento em várias situações. No entanto, criar avaliadores para diagramas com características diversas para cada situação, manualmente, para além de ser trabalhoso, possibilitaria um número limitado de testes.

Tendo isto em conta, decidiu-se criar um gerador automático de grafos conexos. Este gerador está programado para gerar grafos com as características que se pretendam. Para isso, tem os seguintes parâmetros que podem ser alterados quando necessários:

- Número máximo e mínimo de nós
- Número máximo e mínimo de tipos de nós
- Número máximo e mínimo de arcos
- Número máximo e mínimo de tipos de arcos
- Número máximo e mínimo de propriedades por objeto (nós ou arcos)

- Número máximo e mínimo de tipos de propriedades

Configurando estes parâmetros, o gerador encontra um número aleatório nesse intervalo e cria esse número dos elementos correspondentes, para cada um dos pontos descritos acima.

O primeiro passo é a criação de nós. Seja max_n e min_n o número máximo e mínimo de nós do grafo, é escolhido um valor aleatório k tal que $min\_n \leq k \leq max\_n$. Então, em seguida procede-se à criação desses nós. Com os nós são criados também os seus tipos e propriedades, tendo sempre em conta o intervalo em que podem ser representados.

Para a criação de arcos o processo é semelhante, porém contempla ainda a escolha aleatória dos nós de origem e destino. Como esta escolha é aleatória, no final verifica-se se o grafo é conexo. Se porventura não for, são adicionadas o mínimo de ligações possíveis para o tornar conexo.

Cada uma destas gerações tem associadas a si uma semente (*seed*) que funciona como identificador. Desta forma, se durante a execução de alguns testes forem encontradas algumas situações inesperadas pode-se, através desse valor, voltar a gerar exatamente o mesmo grafo, de modo a se poder depurar o avaliador.

Isto seria suficiente se quiséssemos apenas comparar grafos iguais ou dois grafos completamente aleatórios. No entanto, o que se pretende é comparar dois grafos que sejam semelhantes, que possuam características em comum. Para tanto, criou-se a possibilidade de a partir de um grafo gerado aleatoriamente se gerar um outro grafo igual ao primeiro a menos de diferenças configuráveis pelo utilizador. Neste caso, o gerador precisa de saber:

- Número máximo de diferenças nos nós (dn)
- Número máximo de diferenças nos arcos (da)
- Número máximo de diferenças nas propriedades (dp)

Estes parâmetros servem para criar as diferenças no novo grafo. Inicialmente, o novo grafo é uma cópia exata do primeiro, e as diferenças são aplicadas da seguinte forma. Se o grafo original tem n nós, o novo grafo terá n', sendo que $n - dn \leq n' \leq n + dn$. É neste intervalo que é escolhido o valor de n' de forma aleatória. Se for superior a n, são criados e introduzidos novos nós ao novo grafo. Em situação oposta, são removidos os

nós e os arcos respetivos a esses nós. Analogamente, este processo é aplicado aos arcos e às propriedades. Por fim, no grafo onde são aplicadas as diferenças são baralhadas as listas de nós e de arcos, por forma a não ficarem na mesma ordem do grafo original.

No final, tem-se um par de grafos que são iguais a menos de um conjunto de diferenças. Esse conjunto de diferenças é construído à medida que são introduzidos ou removidos elementos ao novo grafo. Este conjunto é essencial, durante os testes, pois representa o conjunto de diferenças que é expectável que o algoritmo encontre quando comparar os dois grafos produzidos pelo gerador.

4.2 Simulação com Dados Sintéticos

Utilizando as potencialidades do gerador aleatório de grafos conexos foi possível realizar experiências, com elevado número de casos de teste, e analisar o comportamento do algoritmo em situações diferentes.

As subsecções seguintes descrevem as várias experiências e expõem os resultados obtidos. Em cada uma das experiências são executados testes com o número de nós dos grafos a variar entre 1 e 30 sendo que para cada um desses valores foram executados 100 testes. Ou seja, no limite, temos 30 situações de testes, onde cada uma delas executa a comparação de 100 pares distintos de grafos. Temos então a utilização do avaliador por $30 \times 100 = 3000$ vezes.

Em todas as experiências com dados sintéticos foram comparados os resultados obtidos da avaliação com os resultados esperados, fornecidos pelo gerador.

Estas experiências decorreram numa máquina com quatro *cores*, com oito processadores i7-3630QM de 2.40GHz e com 8 GB de RAM.

4.2.1 Simulação 1: Mapeamento Simples *vs.* Mapeamento Otimizado

O primeiro teste comparou o comportamento de dois tipos de mapeamento: o mapeamento simples que testa todas as permutações possíveis de mapeamentos; e o mapeamento otimizado, descrito no capítulo anterior.

A questão a que se pretendeu responder foi a seguinte: Será que é mesmo necessário utilizar uma versão otimizada, ou será suficientemente a utilização de todas as

permutações?

Foram executados vários testes para grafos com um variado número de nós (n). Para cada n foram executados cem testes e a duração, exibida nos gráficos, é a do tempo médio dessas cem execuções. Os resultados obtidos estão apresentados em baixo.

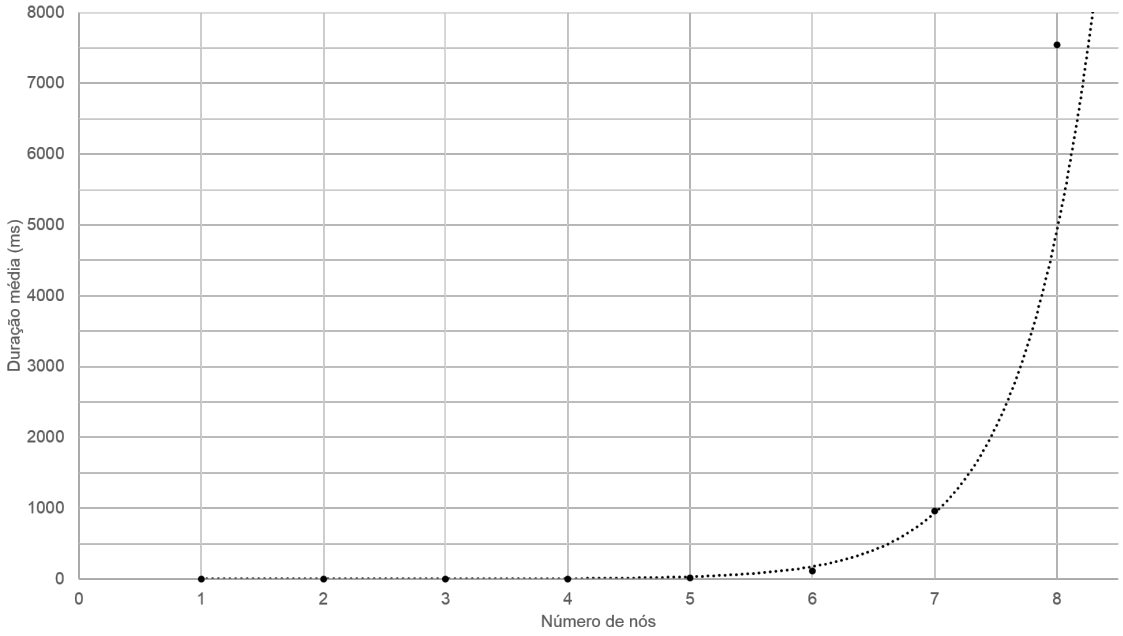

Figura 4.1: Gráfico da duração média de execução em função do número de nós do grafo - Mapeamento simples

Como se pode observar no gráfico da Figura 4.1, quando é utilizado o mapeamento simples, os tempos médios de execução vão crescendo à medida que se aumenta o número de nós, sendo que os aumentos mais notórios dá-se a partir de $n = 5$. Porém, estes dados não são suficientes para se perceber qual o método melhor. A melhor forma de se encontrar as diferenças seria ter as linhas de tendência das duas situações no mesmo gráfico. O motivo para este gráfico não apresentar a curva representativa dos resultados da avaliação com mapeamento otimizado é a escala do eixo do tempo de execução. É difícil definir uma escala que permita compreender as duas curvas em simultâneo, como se vê na Figura 4.2.

Nesta figura destaca-se o número de testes executados para a técnica otimizada. Com o método anterior tinham sido comparados grafos até oito nós, apenas, e neste foram executados testes com n até 30. Observando os tempos de execução na Figura 4.1 vemos que no pior caso ($n = 8$) obteve-se um tempo médio superior a 7 segundos. No

entanto com este novo método, a utilização do critério de corte permitiu obter resultados melhores, tendo o pior caso ($n = 30$) demorado, em média, aproximadamente 6,5 ms.

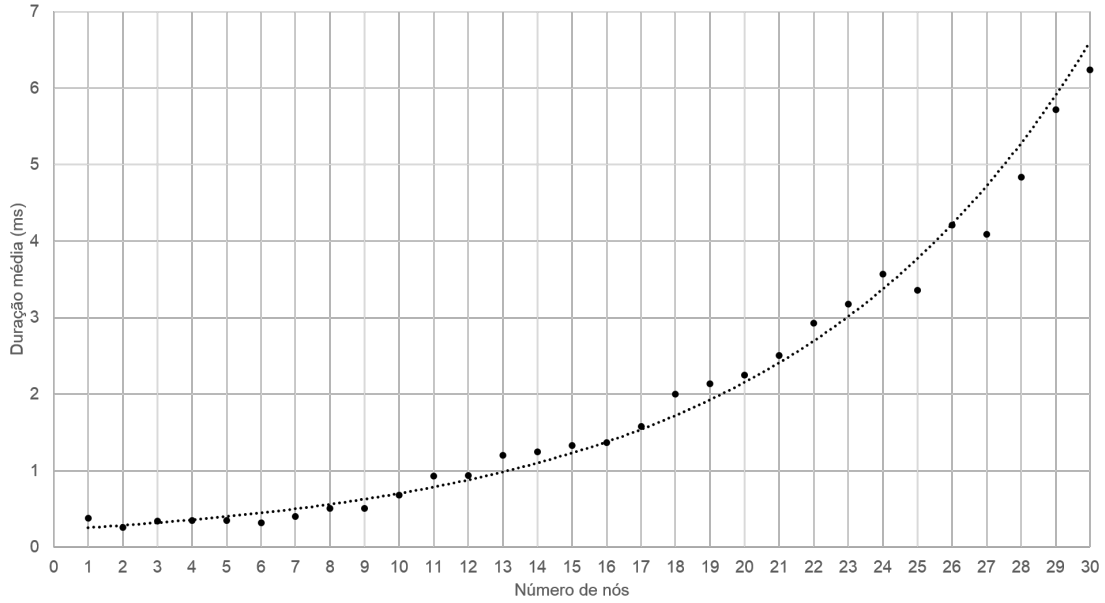

Figura 4.2: Gráfico da duração média de execução em função do número de nós do grafo - Mapeamento otimizado

Com esta experiência, certificou-se o expectável: gerar todas as permutações sem utilização de critérios de corte é muito pouco eficiente comparativamente com o método proposto.

4.2.2 Simulação 2: Analisar o impacto do número de diferenças

Esta experiência pretendeu analisar o impacto do número de diferenças na execução do avaliador. Para isso, executaram-se testes fazendo variar o número de diferenças dos nós.

Foram efetuados 3000 comparações de grafos para cada valor de diferença de nós. Isto é, seja n o número de nós da solução, n' o número de nós da tentativa e d a diferença máxima entre o número de nós solução e tentativa, o que foi feito foi o seguinte:

1. Variar d no conjunto {1, 2, ..., 5}
2. Para cada d, efetuar 100 comparações de grafos para cada n

3. Fazer variar $n : 1 \leq n \leq 30$ tal que $n - d \leq n' \leq n + d$

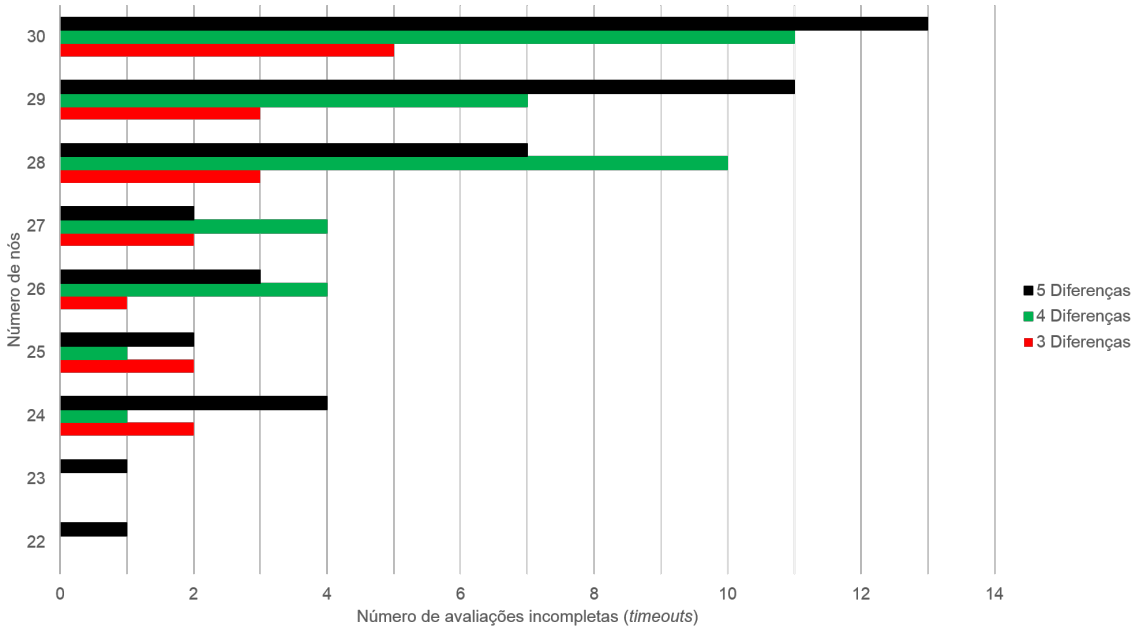

Figura 4.3: Gráfico da duração média de execução em função do número de nós do grafo - Mapeamento otimizado

Neste caso em concreto não é tão relevante avaliar os tempos médios de execução, visto que tal como o esperado vão aumentando à medida que se aumentam o número de diferenças. Aqui, é mais apropriado avaliar o número de *timeouts* (avaliações incompletas) que ocorreram durante todas as avaliações. Esses dados encontram-se sintetizado no gráfico da Figura 4.3.

Cada barra simboliza a experiência realizada para um certo número de diferenças, neste caso 3, 4 e 5. Não estão representadas as barras referentes a uma e duas diferenças, porque em nenhum desses casos foram detetadas avaliações incompletas.

Nas que estão representadas no gráfico, vemos que o *timeout* que ocorre em primeiro lugar diz respeito à experiência com 5 nós de diferença e dá-se para um grafo solução de 22 nós. O que se pode concluir pela análise do gráfico é que a tendência é as avaliações incompletas aumentarem à medida que o número de nós do grafo solução aumenta. Do mesmo ponto de vista, é possível concluir que para grafos solução com o mesmo número de nós, a probabilidade de ocorrer um *timeout* é tanto maior quanto o número de diferença de nós.

Apesar dos *timeouts* detetados, em algumas dessas situações (aproximadamente 50%)

o valor que havia era o esperado, não tendo sido possível certificar-se disso, em tempo útil. A Tabela 4.1 mostra em termos percentuais os dados relativos às avaliações incompletas e as que dessas continham o resultado correto:

Diferenças	*Timeouts*	% *Timeouts*	Corretos	% Corretos
3	18	$\frac{18}{1000} = 1,8\%$	7	$\frac{7}{18} = 38,9\%$
4	38	$\frac{38}{1000} = 3,8\%$	17	$\frac{17}{38} = 44,7\%$
5	44	$\frac{44}{1000} = 4,4\%$	24	$\frac{24}{44} = 54,5\%$

Tabela 4.1: Tabela com dados estatísticos das avaliações incompletas

4.2.3 Simulação 3: Analisar o impacto do número de tipos

A terceira experiência visa analisar a importância dos tipos dos nós numa comparação. Quanto mais tipos existirem é expectável que existam mais nós com características diferentes. Por isso, analisou-se mais uma vez comparando grafos sem diferenças, qual seria o comportamento do algoritmo nos diversos casos.

Para efetuar estes testes limitou-se o gerador a um número máximo de tipo de arcos de 3 e fez-se testes com vários pares de grafos e com diferentes número máximo de tipos de nós: de 1 até 5.

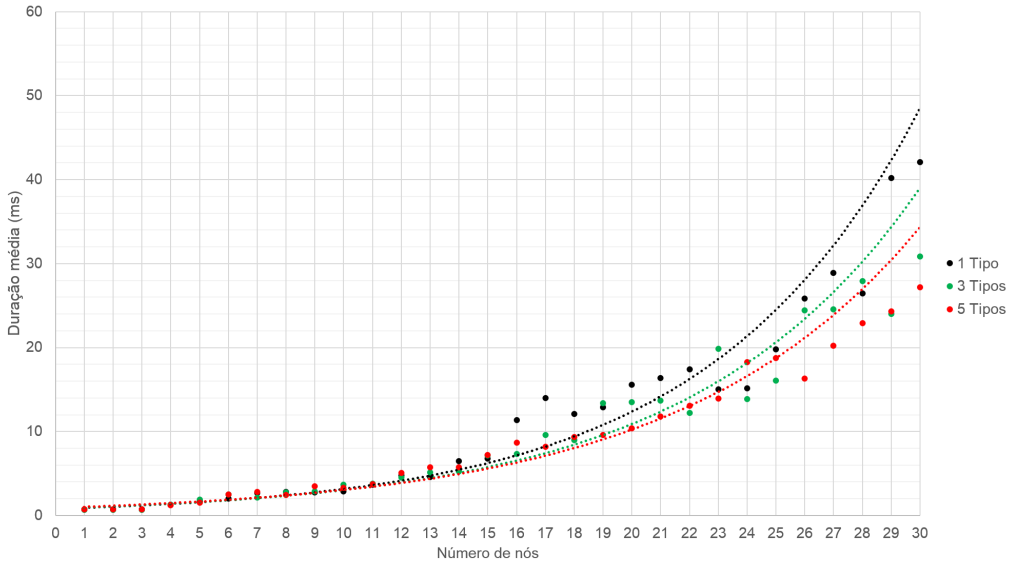

Figura 4.4: Gráfico da duração média de execução em função do número de nós do grafo - Mapeamento otimizado

Na Figura 4.4 pode observar-se que quantos menos tipos existem, maior será o tempo médio de execução. Como se pode observar, no ponto máximo de cada curva (para $n = 30$) temos que para grafos com um só tipo de nós o tempo de execução é, em média, próximo dos 50 ms enquanto que no caso de termos cinco tipos de nós esse tempo baixa para perto dos 30 ms.

4.2.4 Simulação 4: Analisar o impacto do pesos

Relativamente ao critério de corte, importa analisar o impacto que têm os pesos atribuídos ao nós e aos arcos. Nesta experiência, foram executados testes onde são comparados pares de grafos para os casos em que temos os pesos dos nós e dos arcos com os seguintes valores: (50%,50%), (60%,40%), (70%,30%), (80%,20%) e (90%,10%), sendo que o valor do par à esquerda representa o peso dos nós e o da direita o dos arcos.

Decidiu-se realizar estes testes com grafos com algumas diferenças. Para todos estes testes definiu-se que as diferenças entre um grafo e outro seria, no máximo, 3 nós (a mais ou a menos).

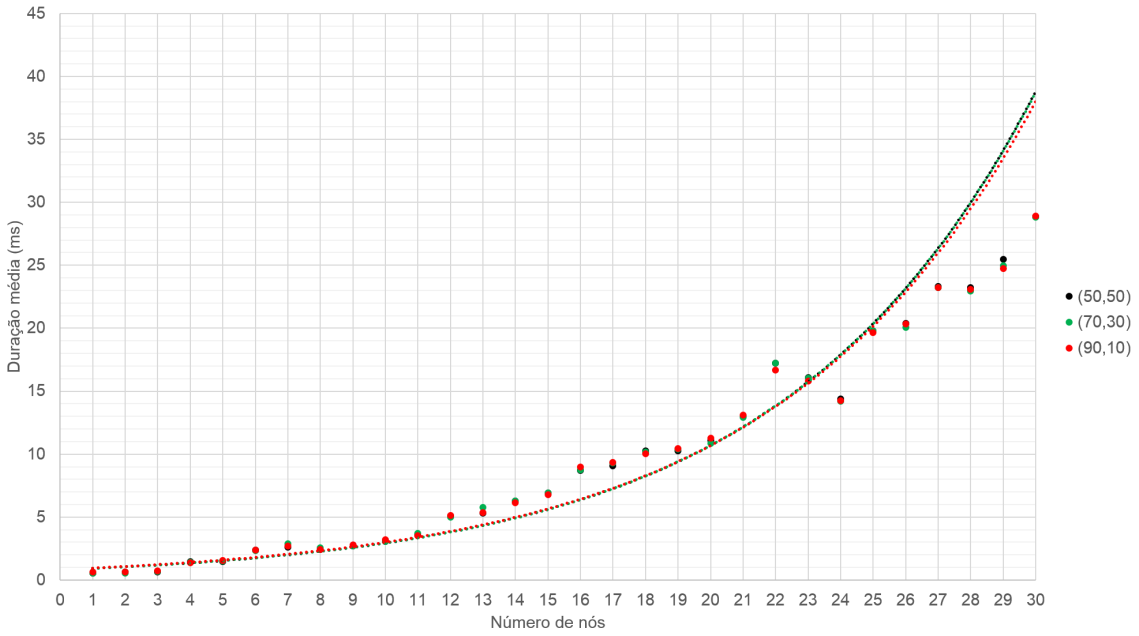

Figura 4.5: Gráfico da duração média de execução em função do número de nós do grafo - Mapeamento otimizado

De todos os casos testados, foram escolhidos para serem representados no gráfico os casos extremos ((50,50) e (90,10)) e o caso médio (70,30). Observando o gráfico da Figura 4.5, observa-se que nas três situações apresentadas, os tempos médios de execução tendem a ser muito semelhantes. Parece que deixa de fazer sentido a importância dada ao facto de o peso dos nós ter de ser superior ao dos arcos. No entanto, existe mais um facto que deve ser analisado e que é apresentado na Tabela 4.2.

Pesos	*Timeouts*
(50,50)	3
(60,40)	1
(70,30)	0
(80,20)	0
(90,10)	0

Tabela 4.2: Tabela de avaliações incompletas por pesos

Esta tabela diz respeito às avaliações que excederam os dois segundos de execução, que é o limite máximo de tempo definido. Relembrando que, para cada par de pesos, os testes executados foram os mesmos, vemos que em três desses testes, com peso dos nós a 50%, o avaliador não conseguiu confirmar se a solução encontrada ao fim de dois segundos é a melhor possível. Do mesmo modo, um desses casos voltou a falhar quando foi testado com o peso dos nós a 60%. Daqui para a frente, vemos que não voltam a surgir casos de avaliações incompletas.

Conclui-se que os pesos não terão grande influência nos tempos médios de execução. Porém, em casos em que seja mais complicado encontrar o mapeamento ideal (devido à existência de alguma indistinguibilidade entre um nó solução e vários da tentativa), quanto maior for o fator de peso da nota dos menos, menor será a probabilidade de o avaliador não conseguir efetuar o corte a tempo.

4.2.5 Conclusões

As experiências realizadas com o gerador de grafos serviram essencialmente para validar o algoritmo. Em todas as experiências, para todos os testes efetuados foi realizada a verificação de que o conjunto de diferenças encontrado pelo avaliador e a nota calculada por ele coincidiam com o esperado. Em todas elas o resultado foi positivo.

Estas experiências permitiram-nos concluir que o algoritmo implementado apresenta, de facto, uma melhoria face ao cálculo de todas as permutações. Pelo menos no que diz respeito a tentativas que estejam relativamente perto da solução. Isto porque estas experiências também permitiram concluir que, à medida que afastamos a tentativa da solução, ou seja, que introduzimos diferenças, é cada vez mais difícil, mais demorado encontrar os valores desejado, e é até mais provável ultrapasse o limite de tempo estabelecido. Isto é semelhante ao que acontece na correção de exercícios feita por pessoas, visto que quanto mais distante está a resposta da solução, mais complicado é para um professor encontrar as semelhanças.

Podemos concluir que uma das formas de tentar atrasar este problema é aumentar o peso dos nós, tendo em conta que, no geral, a aplicação do algoritmo não ficará acelerada mas, para casos específicos, torna a probabilidade de *timeout* menor.

Por outro lado, vimos que quantos mais tipos de nós existirem, mais rápido é o cálculo da comparação. Isto deve-se ao facto de que, quando existem mais tipos, a probabilidade de existirem mais nós diferentes é maior, e quanto mais nós com características distintas existirem, mais fácil é encontrar o mapeamento correto.

Estes resultados levam a crer que ainda haverá espaço para melhorar o algoritmo, nomeadamente, no que às avaliações incompletas diz respeito. Pensa-se que ainda será possível afinar melhor o critério de corte ou a forma como são gerados os mapeamentos, de modo a ser encontrada mais rapidamente a solução desejada. Ou, se tal não for possível, pode ser importante a implementação de um mecanismo que identifique previamente que uma avaliação irá falhar, mesmo antes de se iniciar a gerar mapeamentos.

4.3 Simulação com Alunos

Depois de realizados os testes com dados sintéticos, a questão que se impunha era: Qual será o comportamento do algoritmo com problemas reais e com respostas de alunos.

De forma a responder a esta pergunta estava programada uma experiência com diagramas UML destinada a alunos da unidade curricular de Arquitetura de *Software*. Contudo, devido à recente reformulação do plano de estudo, o número de alunos inscritos nessa disciplina reduziu bastante pelo que foi impraticável a sua realização.

A alternativa que surgiu foi a de avaliar diagramas de modelação EER com os alunos inscritos na unidade curricular de Bases de Dados que são cerca de 160. Esta experiência causou-nos alguma expetativa, uma vez que este tipo de exercícios são avaliados em exame e esta ferramenta poderia ser utilizada pelos alunos para se prepararem para isso.

Pretendia-se avaliar com esta experiência a qualidade do *feedback* produzido. Portanto, decidiu-se que para o mesmo aluno, metade dos exercícios produzem um *feedback* completo, com a percentagem de semelhança e sugestões para melhorar a solução. Enquanto que a outra metade dos exercícios apresenta apenas a percentagem de semelhança.

De forma a realizar a experiência como desejado teve de se escolher um número par de exercícios (seis, neste caso). De forma a nem todos os alunos terem as dicas nos mesmos exercícios, definiu-se que os *feedbacks* completos seriam enviados quando a soma do número mecanográfico do aluno com o número do problema fosse par. Nas outras circunstâncias, o aluno apenas recebe uma informação mínima (o valor da comparação).

Como não foi possível afinar os parâmetros de cálculo da nota, os alunos foram alertados de que a percentagem que surge no ecrã é apenas uma razão de semelhança entre os dois diagramas e não a nota que obteriam se resolvessem o exercício daquela forma, em exame.

4.3.1 Descrição do sistema

Para a realização desta experiência foi necessária a criação de uma expansão do *Dia* para suportar os elementos dos modelos EER, visto que de origem, apenas existem os elementos ER.

De seguida, com a ajuda do professor regente de Bases de Dados, foi possível recolher alguns exercícios deste género, que foram alvo de avaliação em anos anteriores e as respetivas soluções. O passo seguinte foi criar o modelo da solução utilizando o *Dia*.

Estando reunida a informação necessária para a experiência, faltava integrar o avaliador com um sistema que os alunos pudessem utilizar. A escolha recaiu sobre o *Mooshak* que é utilizado no departamento para as avaliações automáticas de exercícios de programação e consultas SQL.

O resultado da integração do avaliador no *Mooshak* foi o apresentado na Figura 4.6.

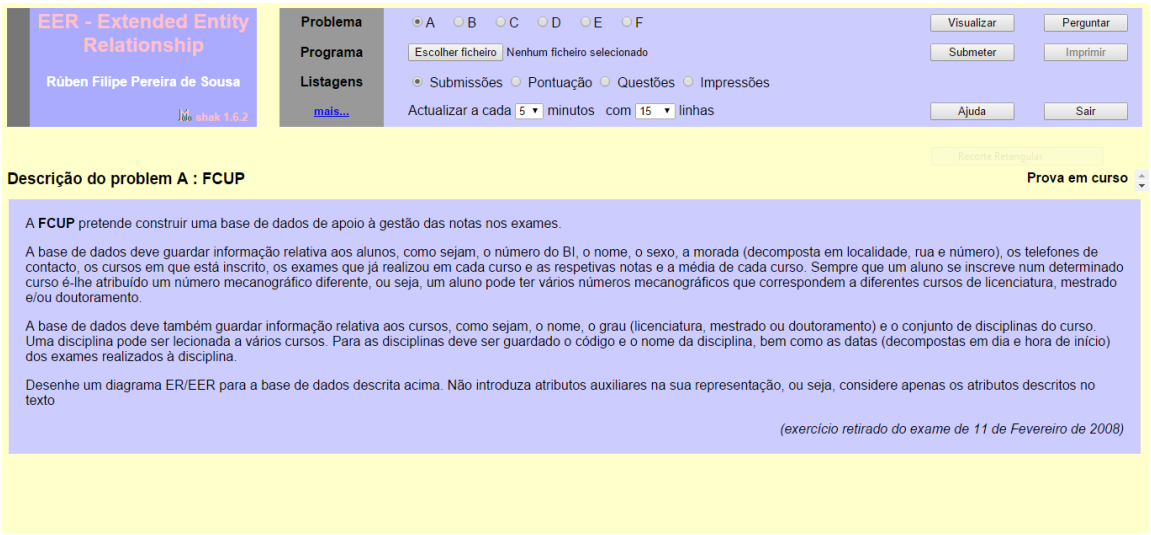

Figura 4.6: Interface gráfica onde está integrado o avaliador

Este ecrã permite a visualização do problema, a troca de exercício e a submissão de respostas, entre outras coisas. A cada submissão, o sistema envia automaticamente a página com o resultado da avaliação.

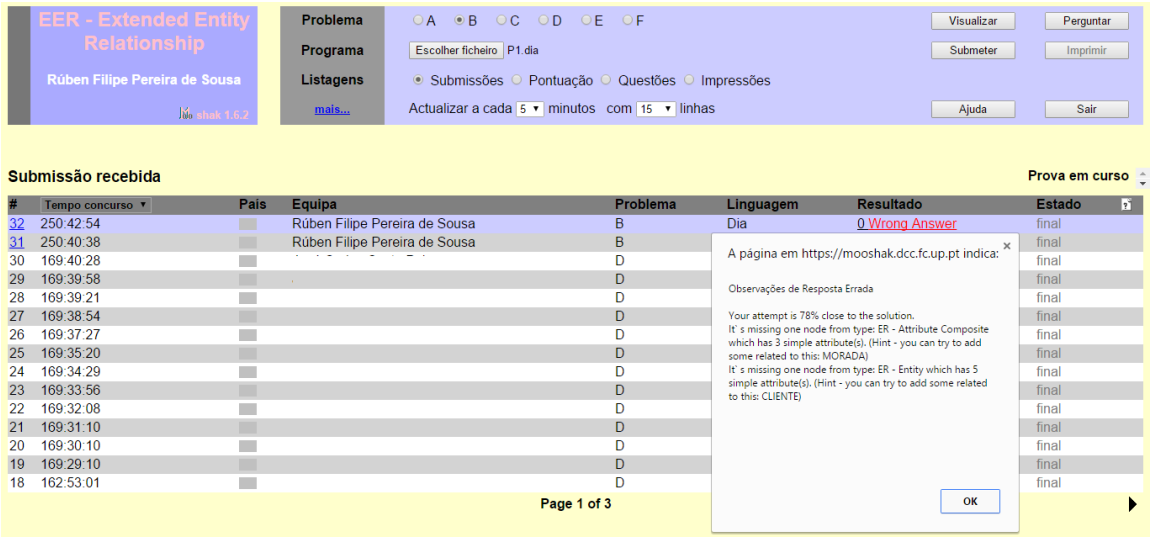

Figura 4.7: Página de resultado da avaliação

Caso a avaliação esteja correta, surgirá, a verde, ao lado da indicação do problema resolvido, a frase "*Accepted*". Caso contrário, aparecerá, a vermelho, "*Wrong Answer*",

tal como na Figura 4.7. Carregando nessa frase, abre uma pequena janela onde é apresentado o *feedback*.

4.3.2 Conclusões

Apesar das elevadas expetativas nesta experiência, a adesão por parte dos alunos foi muito baixa. Esse facto não permitiu a realização de uma análise quantitativa do efeito do *feedback*. No entanto, as poucas participações que tivemos foram dando algumas sugestões sobre como melhorar as mensagens de *feedback*.

A primeira sugestão prendeu-se com o tamanho das frases. Inicialmente as mensagens enviadas continham muito texto, quando a informação essencial podia ser bastante sintetizada. Então alterou-se o formato das mensagens para atender a essa sugestão.

A segunda recomendação fornecida pelos alunos foi relativamente ao número de sugestões apresentadas. Com a avaliação eram detetadas todas as diferenças, eram apresentadas uma mensagem por cada diferença encontrada. O argumento utilizado é que, por vezes, umas diferenças estão relacionadas com outras. Por exemplo, se faltar uma entidade faltarão, também todos os atributos que lhes estão relacionados e todas as suas ligações. Então, como todas estas diferenças eram detetadas, era enviado uma mensagem que sugeria a inserção de cada um desses elementos. Porém, se fosse indicado apenas que faltava uma entidade, por norma, o aluno construiria os atributos e as ligações associadas, ou pelo menos, parte deles. Este argumento pareceu-nos válido, pelo que optou-se por passar a enviar um *feedback* por nível de importância. Isto é, se existirem diferenças ao nível dos nós, apenas são dadas sugestões sobre os nós. Caso contrário, se existirem diferenças ao nível dos arcos, apenas sugestões sobre isso serão dadas. E por último, apenas quando os nós e arcos estão em quantidades apropriadas é que é dado o *feedback* em relação aos atributos.

Associado ao elevado número de sugestões na mensagem, em casos extremos onde existe um elevado número de diferenças a mostrar, a percentagem de semelhança entre a solução e a tentativa era cortada, visto que era a última linha a ser impressa. Outra alteração que se realizou foi a passagem desse valor do final para o início. Assim, certifica-se de que surgirá sempre.

Por fim, uma conclusão que foi possível retirar decorrente de alguns exercícios foi que, em alguns casos, indicar o apenas do tipo do nó em falta poderia não ser suficiente. Visto que poderia levar a que um aluno fosse adicionar um nó do mesmo tipo mas com

um significado completamente diferente, afastando, assim, ainda mais a sua resposta da solução. A solução encontrada foi, juntamente com o tipo do nó, apresentar como sugestão o nome que esse nó contém na solução. Assim o aluno compreenderá melhor o contexto em que se insere o nó em falta e encaminhará a sua resposta na direção correta.

Após aplicarmos todas estas alterações, foi possível perceber que existiu alguma melhoria nas respostas obtidas. Contudo, como o número é bastante reduzido não podem ser tiradas conclusões significativas.

Capítulo 5

Conclusões

5.1 Trabalho Desenvolvido

Durante esta dissertação de mestrado foi possível atingir o principal objetivo: criar um avaliador de diagramas de diferentes tipos. Para isso foram atingidas várias metas que haviam sido propostos inicialmente: a criação do algoritmo, do conversor de diagramas em grafos, do afinador de parâmetros e a execução testes de validação.

Apesar disso, o mais importante foi a implementação do algoritmo de comparação de grafos. Este algoritmo foi implementado de forma genérica, tal que consegue ser utilizado para comparar dois diagramas, independentemente da linguagem utilizada.

Isto é possível devido à utilização de um conversor que suporta de momento três tipos de diagramas (UML (diagrama de classes e de casos de uso) e EER) em grafos estendidos prontos a serem comparados.

Os testes executados, quer com dados artificiais quer com problemas reais, demonstram que o algoritmo funciona de forma satisfatória no que diz respeito à procura de diferenças e cálculo da nota e até aos tempos de execução. Porém, existem as situações em que o avaliador não consegue encontrar o desejado antes de atingir o limite de tempo. Esse é o ponto mais fraco desta abordagem. Contudo, este trabalho tem condições para ser adaptado e melhorado, de forma a que no futuro os resultados sejam ainda mais promissores.

A submissão de um artigo [LS] descrevendo esta ideia e a sua respetiva apresentação no *Sysmposium on Languanges, Applications and Technologies* (SLATE), em Madrid,

no mês de Junho de 2015, foi muito bem acolhida pela comunidade, que se mostrou curiosa em saber mais novidades sobre a evolução do projeto.

5.2 Trabalho Futuro

Apesar dos resultados obtidos no âmbito deste projeto, há ainda um caminho a percorrer. Desde logo as melhorias sugeridas em secções anteriores. Tendo em conta que o ponto fraco deste projeto são os casos em que o avaliador não consegue terminar a execução em tempo útil, deve-se focar agora neste ponto. Existem duas soluções. Idealmente, deveria ser feita uma afinação do algoritmo de comparação de nós e geração de mapeamentos, de forma a que por muitos nós indistinguíveis que existam entre solução e tentativa, seja possível identificar o mais cedo possível as melhores correspondência, permitindo assim o alcance do critério de corte mais cedo. No entanto, neste momento, não se está em condições de se afirmar que tal é possível. Por isso, a outra alternativa é a criação de um mecanismo que, antes de iniciar a comparação dos grafos, possa, com recurso a uma heurística, prever se a avaliação vai falhar ou não. No caso de ser detetado que falharia, não executaria o algoritmo.

Para além disto, está ainda pendente uma melhor validação com alunos. Isto porque, apesar de se ter aberto a plataforma a um elevado número de alunos, as respostas que obtivemos foram em número muito reduzido. Portanto, no futuro uma das tarefas a executar terá de ser a repetição desta experiência.

Contudo, este projeto pretende evoluir num outro sentido. O próximo passo é a utilização deste algoritmo para a avaliação de exercícios de programação, possibilitando a localização de erros no código fonte. Quem está a aprender a programar, comete frequentemente erros. Contudo, o que os avaliadores automáticos fazem, por norma, é apenas indicar se a resposta está correta ou errada. No limite podem indicar a percentagem de casos de teste que passou, apresentando alguns exemplos que falharam. O objetivo é, nas situações em que a resposta não é marcada como correta, possibilitar-se uma análise de código fonte que permita encontrar o erro.

Para isso, pretende-se que o sistema esteja preparado para comparar a resposta do aluno quer com a solução colocada pelo professor, quer com todas as outras respostas dos alunos (que sejam diferentes da apresentada pelo professor) que tenham sido marcadas como certas.

Para isso, é necessário construir um grafo a partir do código fonte. Esse grafo é um

grafo semântico abstrato, isto é, uma árvore sintática onde os nomes de variáveis e sub-programas são substituídos por arcos. Aí, teríamos que os tipos de nós poderiam ser algo como:

- Variável
- Funções
- Instruções de salto (*branch*)
- Operações aritméticas
- Controlo de fluxo
- ...

Quanto às propriedades podem estar relacionadas com o tipo de variável ou de retorno de função ("*int*", "*boolean*", "*void*", etc.) ou com o seu valor ("1", "true", etc.). Mais uma vez, o objetivo é a abstração dos nomes visto que nem todos os programadores utilizam as mesmas regras de nomenclatura de variáveis.

Vejamos os seguintes exemplos que pretendem implementar uma função para determinar se um certo número é primo ou não. A vermelho estão destacados os erros de cada uma das implementações (ver Figura 5.1).

```
boolean isPrime(int n) {
for(int d=1; d<sqrt(n); d++)
       if(n % d == 0)
           return false;
   return true;
}
```

```
boolean prime(int k) {
boolean prime = true;
int i = 1;
while(i < k) {
       if(k % i == 0)
           prime = false;
       i++;
   }
   return prime;
}
```

Figura 5.1: Duas tentativas (com erros) de implementar uma função

Como se pode ver, as duas implementações são diferentes mas ambas estão incorretas. Nos dois casos, começam o teste por verificar se um número é divisível por 1. Como todos os números o são, o valor retornado será sempre *false*. Ou seja, em ambos os casos, o valor pelo qual começam o teste de divisibilidade deveria ser 2. No caso da

esquerda, existe ainda o erro de, no ciclo iterativo, o limite máximo estar definido de forma errada. Neste caso, onde está "menor" deveria estar "menor ou igual".

Este avaliador surge com o intuito de se criar um mecanismo que permita aos alunos de Introdução à Programação, desenvolverem as suas capacidades enquanto estudam. Estas mensagens com os seus erros seriam úteis para os ajudar a ultrapassar situações de erro. Isto pode ser uma grande ajuda em *Massive Online Open Courses* (MOOC) de programação, colmatando a falta de acompanhamento de um professor.

Referências

[ASI07] Noraida Haji Ali, Zarina Shukur, and Sufian Idris. A design of an assessment system for uml class diagram. In *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on*, pages 539–546. IEEE, 2007.

[CCP04] Virginio Cantoni, Massimo Cellario, and Marco Porta. Perspectives and challenges in e-learning: towards natural interaction paradigms. *Journal of Visual Languages and Computing*, 15:333–345, 2004.

[Dia] Dia diagram editor. http://dia-installer.de/. Acedido: 2015-06-10.

[DLO05] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.

[DOM] Domjudge team manual. http://www.domjudge.org/docs/team-manual.pdf/. Acedido: 2015-12-04.

[dQ12] Ricardo Alexandre Peixoto de Queirós. *A framework for practice-based learning applied to computer programming.* 2012.

[GP05] Paul Gross and Kris Powers. Evaluating assessments of novice programming environments. In *Proceedings of the first international workshop on Computing education research*, pages 99–110. ACM, 2005.

[Jay10] Ambikesh Jayal. *Framework to Manage Labels for E-assessment of Diagrams.* 2010.

[Jen02] Tony Jenkins. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58, 2002.

[LS] José Paulo Leal and Ruben Sousa. A structural approach to assess graph-based exercises. *Sysmposium on Languanges, Applications and Technologies (SLATE)*.

[LS03] José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software—Practice and Experience*, 33(6):567–581, 2003.

[Nic08] Mark Nichols. *E-Primer Series – E-Learning in Context*. Laidlaw College, 2008.

[Pac10] Pedro Xavier Pacheco. *Computer-Based Assessment System for e-Learning applied to Programming Education*. 2010.

[SBP+10] Josep Soler, I Boada, F Prados, J Poch, and R Fabregat. A web-based e-learning tool for uml class diagrams. In *Education Engineering (EDUCON), 2010 IEEE*, pages 973–979. IEEE, 2010.

[SHP+06] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K Hollingsworth, and Nelson Padua-Perez. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. *ACM SIGCSE Bulletin*, 38(3):13–17, 2006.

[SM08] Zarina Shukur and Nurul F Mohamed. The design of adat: A tool for assessing automata-based assignments. *Journal of Computer Science*, 4(5):415, 2008.

[TGP05] Guy Tremblay, Frédéric Guérin, and Anne Pons. A generic and extensible tool for marking programming assignments. In *IASTED Intl conf. on Educ. and Tech*, pages 55–60, 2005.

[TWS12] Pete Thomas, Kevin Waugh, and Neil Smith. Automatically assessing free-form diagrams in e-assessment systems. In *STEM Annual Conference*, 2012.

[VP14] Vinay Vachharajani and Jyoti Pareek. A proposed architecture for automated assessment of use case diagrams. *International Journal of Computer Applications*, 108(4), 2014.

[WG05] Jeremy B Williams and Michael Goldberg. The evolution of e-learning. *Proceedings of Australasian Society for Computers in Learning in Tertiary Education Queensland University of Technology, Brisbane, Australia. Retrieved November*, 17:2011, 2005.

[Zha03] Dongsong Zhang. Powering e-learning in the new millennium: An overview of e-learning and enabling technology. *Information Systems Frontiers*, 5(2):201–212, 2003.